Physics-Aware Convolutional Neural Networks for Computational Fluid Dynamics

Viktor H. Grimm

Physics-Aware Convolutional Neural Networks for Computational Fluid Dynamics

Inaugural-Dissertation

zur

Erlangung des Doktorgrades

der Mathematisch-Naturwissenschaftlichen Fakultät

der Universität zu Köln

vorgelegt von

Viktor Hermann Grimm

aus Troisdorf

Köln, 2023

Berichterstatter:	Prof. Dr. Axel Klawonn (Universität zu Köln) Prof. Dr. Oliver Rheinbach (TU Bergakademie Freiberg)
Datum der Disputation	10. Juli 2023

Abstract

Determining the behavior of fluids is of interest in many fields. In this work, we focus on incompressible, viscous, Newtonian fluids, which are well described by the incompressible Navier-Stokes equations. A common approach to solve them approximately is to perform Computational Fluid Dynamics (CFD) simulations. However, CFD simulations are very expensive and must be repeated if the geometry changes even slightly.

We consider Convolutional Neural Networks (CNNs) as surrogate models for CFD simulations for various geometries. This can also be considered as operator learning. Typically, these models are trained on images of high-fidelity simulation results. The generation of this high-fidelity training data is expensive, and a fully data-driven approach usually requires a large data set. Therefore, we are interested in training a CNN in the absence of abundant training data. To this end, we leverage the underlying physics in the form of the governing equations to construct physical constraints that we then use to train a CNN.

In particular, we identify the output of a CNN with grid functions on a uniform, tensor-product grid and use finite difference stencils to approximate the necessary derivatives in the residuals of the governing equations. We formulate a loss function by, for example, taking the mean squared sum of these residuals. We are then able to train a CNN by minimizing this loss function with regards to the networks parameters. Thus, we require no reference data to train the CNN. We refer to this as the physics-aware approach and to a CNN trained in this way as a physics-aware CNN.

This approach is based on the finite difference method. In fact, the physics-aware loss function is a least-squares formulation of the nonlinear system of equations obtained by discretizing the governing equations using the finite difference method. As such, it is possible to encounter similar difficulties as with the finite difference method. In this work, we adopt solution ideas for these problems from the literature on the finite difference method applied to the incompressible Navier–Stokes equations and modify our approach to accommodate them. In particular, among others, we use higher-order finite difference schemes, a pressure stabilization, and upwind schemes.

We present results for various model problems, including two- and three-dimensional flow in channels around obstacles of various sizes and in non-rectangular geometries, especially arteries and aneurysms. We compare our novel physics-aware approach to the state-of-the-art data-based approach and also to a combination of the two, a combined or hybrid approach. In addition, we present results for an extension of our approach to include variations in the boundary conditions.

Acknowledgements

I would like to thank my advisor Professor Axel Klawonn for the opportunity of being a part of his team and being able to work with him on this project. I am very grateful for the support and guidance I received under his supervision. Furthermore, I would like to take this opportunity to thank Professor Oliver Rheinbach for reviewing my thesis.

My special thanks go to Alexander Heinlein for his support and advice throughout my work. The many exciting discussions and constant feedback were very helpful. My thanks are also to Achim Basermann for his mentoring and encouragement.

I would also like to thank all my present and former colleagues who supported me on a daily basis. In alphabetical order, I express my gratitude to Christian Hochmuth, Jascha Knepper, Natalie Kubicki, Martin Lanser, Lucas Mager, Sabine Musielack-Erle, Lea Saßmannshausen, Matthias Uran, Adam Wasiak, and Janine Weber. Many thanks for the pleasant and collegial working atmosphere, for the many wonderful and fruitful discussions. At this point I would also like to thank Martin Kühn for his early support and motivation during my bachelor and master studies.

I would also like to thank my family and especially my brother, Robert Grimm, who read and commented on my entire thesis and was always there to give me advice.

I would especially like to thank my girlfriend Jana Frey, who supported me emotionally throughout the whole period. Not only through the good times, but also through the hard times, especially during the extremely stressful lockdown periods, when she put up with me in our one-room apartment.

This work was performed as part of the Helmholtz School for Data Science in Life, Earth and Energy (HDS-LEE) and received funding from the Helmholtz Association of German Research Centers.

I gratefully acknowledge the use of the computational facilities of the Center for Data and Simulation Science (CDS) at the University of Cologne as well as of the Department of Mathematics and Computer Science of the Technische Universität Bergakademie Freiberg operated by the University Computing Center (URZ) and funded under grant application No. 100376434 to the State Ministry for Higher Education, Research and the Arts (SMWK) of the Federal State of Saxony on Artificial Intelligence and Robotics for GeoEnvironmental Modeling and Monitoring.

Contents

List of Figures

List of Tables

1 Introduction

Determining the behavior of fluids is of interest in many fields of engineering and science. In civil engineering, for example, knowledge of fluid flow behavior is needed to design water supply, drainage, or irrigation systems; see, for example, [111]. Mechanical engineers design pumps, turbines, and fans in hydraulic systems; cf. [194]. In all of these applications, simulations are typically performed in advance to determine ideal geometries and flow conditions. Medical applications are of particular interest because of their impact on human life. For example, in the study of aneurysms, it is of interest to know the flow behavior of blood in and around an aneurysm to assess whether the aneurysm is likely to rupture and surgical intervention is warranted.

As a motivational example, let us take a closer look at intracranial aneurysms and their relevance. Intracranial aneurysms have a prevalence of about 1 - 5% [18] and when they rupture, about 60% of patients die immediately after the rupture [32]. Ruptured aneurysms are treated by surgical clipping and endovascular coiling [144]. However, with the increasing accessibility of medical imaging, the incidental finding of unruptured intracranial aneurysms during routine medical imaging is becoming more common [186]. Unruptured intracranial aneurysms can be monitored, for example, by regularly Computed Tomography Angiography (CTA) or Magnetic Resonance Angiography (MRA) [196] of the brain [182]. If the aneurysm is likely to rupture and preventive surgery is therefore necessary, patients may be treated with coiling [17], which involves inserting platinum coils into the aneurysm through a catheter; stenting [56], which requires inserting a metal

or plastic tube to stabilize the artery; or clipping [161], which is the practice of attaching a metal alloy, most recently titanium, clip to the base of the aneurysm, thus cutting off the blood flow. Before treatment, the need for surgical intervention must be determined. This generally depends on the size and shape of the aneurysm [120] and correlates with certain characteristics of the patients, such as age, alcohol, and caffeine consumption, as well as general health [59].

There is a relationship between the flow field and, in part, the resulting Wall Shear Stress (WSS) and the risk of rupture of intracranial aneurysms [209]. Both low and high WSS have been associated with intracranial aneurysm growth [5] and their subsequent rupture. It has been shown, however, that the flow measurements that can be made by CTA or MRA are largely inaccurate; see, e.g., [48]. As an alternative to CTA or MRA measurements, it is possible to use Computational Fluid Dynamics (CFD) simulations to determine flow behavior and compute relevant key parameters, such as the WSS, to determine the risk of rupture [107]. It is also possible to use a combination of CFD simulations to determine the flow behavior and a Machine Learning (ML) classifier to assess the risk of rupture for an unruptured aneurysm [2]. However, this approach, and CFD simulations in general, are very time consuming. Let us discuss reasons for this.

The behavior of fluids is determined by their governing equations, which in many cases are the Navier–Stokes equations. Classically, this system of partial differential equations (PDEs) is solved discretely, for example, using the Finite Difference Method (FDM) [101], the Finite Element Method (FEM) [15], or the Finite Volume Method (FVM) [31]. This approach is part of the field of CFD; see, for example, [3, 13, 40, 92, 141, 195]. However, the computational effort required to solve a suitable CFD simulation can be enormous, not only because a fine mesh of the computational domain is usually required. Simulations are especially time-consuming for complex flows, due to, e.g., complex geometries or turbulence. This is particularly problematic because such simulations must be repeated if anything about the setup changes – such as geometry, boundary conditions, or other

parameters. Even a small change can be significant, see, e.g., [37, 167].

For the medical application we are interested in, a single CFD simulation is already too time-consuming, especially when the patient with the aneurysm needs immediate surgery. In addition, there are usually some uncertainties in, for example, the flow velocity of the blood or the exact representation of the geometry, as an MRI or CT scan is not an exact representation of the geometry. We can incorporate these uncertainties using Uncertainty Quantification (UQ) methods; see, for example, [96, 171]. But this usually requires many simulations to be run, which increases the time required. Therefore, it is desirable to develop and use a surrogate model that can be quickly evaluated and that predicts the flow with sufficient accuracy. However, we do not attempt to match the accuracy of a high-fidelity CFD simulation, nor is it necessary. There are a variety of possible surrogate models for CFD simulations, including reduced order models [200], reduced basis models [68], and neural networks [35, 36, 58]. The use of neural networks as surrogate models for CFD simulations can be seen as part of Scientific Machine Learning (SciML) [6, 181] – a new and rapidly developing field of research in artificial intelligence. This field combines scientific computing and machine learning techniques to produce more accurate and interpretable algorithms.

In this work, we focus on Convolutional Neural Networks (CNNs) [97] as surrogate models for CFD simulations. If we consider intracranial aneurysms, we are faced with the difficulty that our knowledge of the geometry is limited. In practice, the geometry of an aneurysm and adjacent arteries in a living patient can only be determined by medical imaging techniques such as CT or MRI scans. These provide us with pixel or voxel images of the relevant geometries. These images may still need to be cleaned of noise [88, 100] and converted to a higher resolution [24, 139], a technique referred to as super-resolution imaging. Both are tasks that can be performed with ML. This provides us with cleaner and more detailed information about the geometry, which is a prerequisite for CFD simulations of adequate quality. CNNs specialize in processing image data and

are therefore directly suitable as surrogate models for CFD simulations in this case. In this thesis, we consider CNNs as surrogate models that take pixel or voxel images of the geometry as input and produce pixel or voxel images of the velocity and pressure of the Navier–Stokes equations as output [35, 36, 58, 152, 160, 178].

There are a number of advantages to using neural networks as surrogate models compared to numerical simulations, including: the ability to make predictions quickly, often orders of magnitude faster than numerical simulations; the ability to generalize to previously unseen input, whereas numerical simulations must be recalibrated and recomputed; and reduced computational cost, as the training and evaluation of neural networks can be efficiently accelerated using, for example, GPUs, whereas high-fidelity numerical simulations often require high-performance computing (HPC) clusters. It should be noted that training a neural network usually takes a very long time. However, since the training takes place in an offline phase and can be done in advance, the length of the training is not particularly relevant for the use case. What is more important is the evaluation of the network, which usually takes only a few seconds or even milliseconds. The speedup achieved by CNNs over CFD simulations has been reported to be on the order of $O(10^2)$ to $O(10^4)$; see, for example, [35, 36, 58].

Of course there are also a number of disadvantages. Typically, a neural network as a surrogate model is trained on very large data sets, which, in our case, requires a large number of CFD simulations. These data sets are very expensive to generate and, in three dimensions, very memory intensive. Furthermore, the optimization of neural networks is a highly non-convex optimization problem, where convergence to the global minimum is a challenging task. There is also the problem of overfitting of the model on the training data set [204], that is, good performance on the training data but worse performance on previously unseen data. Data-driven optimization methods can converge to local minima and lead to models with limited generalization capabilities or predictions that violate existing knowledge, such as physical laws.

To address some of these drawbacks, we incorporate prior knowledge into the CNN training process. This concept of using prior knowledge, often known physical laws, to train an ML model is known as physics-informed ML; see, for example, [80, 125, 189, 201]. This is not to be confused with Physics-Informed Neural Networks (PINNs) [142, 143], a specific method of constructing a physics-based loss function through clever use of Automatic Differentiation (AD) [7], classically for Dense Neural Networks (DNNs), within physics-informed ML. Physics-informed ML can be seen as a subfield of SciML.

In the novel approach presented in this thesis, we use techniques from the finite difference method to generate a physics–based loss function, which we then use to train a CNN as a surrogate model for CFD simulations of an incompressible fluid. Using finite differences combined with CNNs has been applied in many areas [10, 38, 149, 163, 168, 190, 207], including the Navier–Stokes equations [43]. However, most, if not all, previous methods do not account for changes in geometry, or rely on a parameterization of the geometry, and are therefore unable to generalize to previously unseen geometries. The method we propose is able to predict the flow field and pressure in varying geometries without having been trained on reference data.

In particular, we identify the pixel images on which we train the CNN to be uniform meshes – and the pixel images of the solution variables that the CNN produces as outputs, i.e., the predictions, to be grid functions defined on said uniform meshes. We then use finite difference discretizations to approximate the partial derivatives of the solution variables in the governing equations, and obtain pixel-wise residuals of the Navier–Stokes equations. We want to minimize these residuals to obtain physically correct predictions. To this end, we formulate a loss function over the residuals, e.g., by computing the mean squared sum, and train our CNN by minimizing this physics-aware loss function.

Overall, in this thesis we present a method to train CNNs as surrogate models for CFD simulations in varying geometries using the governing equations. In particular, this method does not require any reference data for training. However, in the presence of

training data, it is possible to train a CNN using both the training data and the governing equations. We also present an extension of the underlying surrogate model approach to variations in boundary conditions, using the inflow velocity as an example.

The remainder of this thesis is organized as follows. In the second chapter, we introduce some of the fundamentals from the field of CFD. This includes the incompressible Navier–Stokes equations and the finite difference method. We also discuss some difficulties that may arise in the numerical solution of the incompressible Navier–Stokes equations. We conclude the chapter by defining model problems to test our method with Next, in chapter 3, we give an introduction to deep learning, machine learning, and, in particular, DNNs and CNNs. Moreover, we discuss some difficulties that may arise in training neural networks. After that, we give an overview of neural networks as surrogate models for CFD simulations in chapter 4. Here, we also focus on DNNs and CNNs. In chapter 5, we describe the procedure for constructing a physics-aware CNN, the primary method of this thesis. Then, in chapter 6, numerical results based on two-dimensional geometries are presented. Here, we test physics-aware CNNs in detail and compare them with data-based CNNs and combined, or hybrid, CNNs that are trained on a combination of the physics-aware and data-based loss functions. Subsequently, in chapter 7, we present three-dimensional results. In particular, we address the limitations of our approach in three dimensions. Finally, we summarize the main observations made in this thesis and present an outlook for future work in chapter 8.

2 Computational Fluid Dynamics

In general, we are interested in finding numerical solutions for a differential equation. In this thesis, we are particularly interested in finding a numerical solution for the incompressible Navier–Stokes equations – or, more specifically, to use existing numerical solution methods to develop an adequate deep learning-based surrogate model. To achieve this, we need some preparatory work, which we cover in this chapter. We begin by briefly describing the governing equations of incompressible fluids in section 2.1. We then move on to the field of Computational Fluid Dynamics (CFD), which encompasses the desired numerical methods, in section 2.2. In particular, we address the Finite Difference Method (FDM) in section 2.2.2 and address some rather impactful problems of this method with respect to the incompressible Navier–Stokes equations in section 2.2.3. For a more detailed discussion of CFD in general, we refer the reader to [3, 13, 40, 92, 141, 195]. In this chapter, we limit ourselves to two-dimensional geometries for the sake of simplicity. However, extensions to three dimensions are possible without difficulty.

2.1 Navier–Stokes Equations for Incompressible Fluids

The Navier–Stokes equations are partial differential equations that describe the behavior of viscous fluids. They are derived from the basic principles of continuity of momentum and mass, and sometimes also energy. For such a derivation, we refer to [153, Sect. 2.4] and the references therein. The Navier–Stokes equations can be expressed in several different forms, especially in two dimensions. The most common of these is the formulation

in primitive variables, which is expressed with the help of a velocity vector field and a pressure field, to which we restrict ourselves in this work.

The stationary Navier–Stokes equations for incompressible fluids are given by

$$(\vec{u} \cdot \nabla)\vec{u} - \nu\Delta\vec{u} + \nabla p = 0 \quad \text{in } \Omega, \tag{2.1}$$

$$\nabla \cdot \vec{u} = 0 \quad \text{in } \Omega, \tag{2.2}$$

where $\vec{u} = (u, v)^T$ is the velocity vector, p is the pressure, ν is the kinematic viscosity, ∇ is the gradient operator, Δ is the Laplacian, and $\nabla\cdot$ is the divergence operator. Here we have divided the equation by the density, so p is actually the kinematic pressure and the relation $p = \tilde{p}/\rho$ holds, where $\tilde{p}$ is the actual, or static, pressure, and ρ is the density. However, in the remainder of this work we refer to the kinematic pressure as the pressure. In the equations we have also neglected any forces, sources, and sinks.

The first equation, eq. (2.1), is a vector equation, consisting of two (or three in three dimensions) component equations. They describe the conservation of momentum; hence we call them momentum equations. Here, $(\vec{u} \cdot \nabla)\,\vec{u}$ is called the convective term, and the term $\nu\Delta\vec{u}$ is called the diffusive term. The second equation, eq. (2.2), is a scalar equation and describes the conservation of mass, and we call it the mass equation. Strictly speaking, only the momentum equations 2.1 are referred to as the Navier–Stokes equations and, in the case of incompressibility, the mass equation 2.2 is referred to as the divergence-free constraint. Nevertheless, in the rest of the work we refer to equations 2.1 and 2.2 together as the Navier–Stokes equations.

Let us note that the primitive-variable formulation we have presented here, called the advective-convective form, is not the only possible one. There are various forms of this formulation that are all equivalent on a continuum, but behave differently when spatially discretized. It is particularly noteworthy that in the derivation of the advective-convective form the condition $\nabla \cdot \vec{u} = 0$ is used. In particular, this implies that a velocity field which, together with a suitable pressure, satisfies eq. (2.1) but not eq. (2.2) may be far from

being a solution of the Navier–Stokes equations. For a detailed overview and discussion of most of the formulations, please refer to [53].

There are a number of quantities used to describe fluid flow. One of them is the Reynolds number. The Reynolds number is a dimensionless quantity that describes the nature of flow. It is defined as the ratio of inertial to viscous forces, or advection to diffusion [140]. The Reynolds number is defined as

$$Re = \frac{U \cdot L}{\nu}, \tag{2.3}$$

where U is the flow velocity, and L is a characteristic length of the geometry under consideration. For example, for a straight tube, L can be defined as the tube diameter. At low Reynolds numbers, the flow is laminar; i.e., viscous forces dominate and the flow is characterized by parallel layers. At higher Reynolds numbers, the flow becomes turbulent; i.e., inertial forces dominate, and vortices, eddies, and other flow instabilities occur. It is not possible to define a clean transition point from laminar to turbulent flow. In general, fluid flow exhibits chaotic behavior, and even slight variations in the boundary geometry – or small perturbations in the flow – can result in significantly different flow patterns. Nevertheless, the Reynolds number is an important indicator because flows with the same Reynolds number behave similarly.

To obtain a well-posed problem, we need to add suitable boundary conditions to the equations. The correct boundary conditions for the various formulations of the incompressible Navier–Stokes equations as well as for the various solution methods have been intensively discussed in the literature in the past; see for example [53, 148]. We only provide boundary conditions for the formulation in primitive variables and do not discuss boundary conditions further but refer to the literature [15, 40, 101, 105, 195].
Using the formulation in primitive variables, it is sufficient to impose the velocity everywhere on the boundary [148]. In order for the pressure to be uniquely defined, we must additionally specify it in at least one point. However, it is also possible to impose a combination of boundary conditions for the velocity and the pressure. Usually, the

domain Ω possesses an inflow boundary $\partial\Omega_{\text{in}}$, where Dirichlet boundary conditions for the velocity are prescribed, and an outflow boundary $\partial\Omega_{\text{out}}$, where we could for example impose Dirichlet boundary conditions for the pressure. It is also possible to prescribe a passive Neumann boundary condition for the fluid flow here, i.e., $\frac{\partial u_n}{\partial n} = 0$, where n is the outward facing normal vector on the outflow boundary; see, for example, [53]. All other boundaries are generally walls $\partial\Omega_{\text{wall}}$ through which no fluid flow is permitted. Commonly, the so-called no-slip condition is applied at walls; i.e., a velocity of zero is prescribed.

2.2 Methods of Computational Fluid Dynamics

The area of Computational Fluid Dynamics (CFD) covers numerical methods for the solution of the governing equations for fluid flow. These are most often the Navier–Stokes equations. Three of the core methods of CFD are the Finite Element Method (FEM) [15, 28], Finite Volume Method (FVM) [31, 122], and the Finite Difference Method (FDM) [101, 170, 174, 179]. All methods involve decomposing the area in which the behavior of the fluid is to be determined into a finite set of subareas or subvolumes, typically referred to as a mesh. They then differ in how the solution is approximated. Regardless of the chosen approximation, all methods result in a system of algebraic equations which, generally, can be solved with a direct or iterative method or nonlinear optimization.

With the finite element method, the problem to be solved is first converted into a variational formulation and then this continuous problem is converted into a discrete problem by restriction to a finite basis. For an overview of the FEM we refer to [15].

The finite volume method involves considering the integral of the equations to be solved over each cell or volume of the mesh and using Gauss's theorem to transform the volume integrals into surface integrals. The solution is then approximated at the cell interfaces in form of fluxes. A key advantage of the FVM is that it is conservative, in that a flux entering a cell via the common face (or edge) with a neighboring cell is identical to the

flux leaving the neighboring cell via the face (or edge) [105]. For an overview of the FVM with regards to hyperbolic equations and the incompressible Navier–Stokes equations, we refer to [102] and [105], respectively.

With the finite difference method, the solution is approximated at either the cell centers or the nodes of the mesh. The differential operators in the differential equations are then approximated by finite differences. Later in this thesis, we combine elements of this method with machine learning and therefore we discuss it in more detail below.

Throughout this chapter, let $\Omega = [0,1] \times [0,1]$ be the unit square and

$$\Omega_h = \{x_{i,j} \mid 0 \leq i, j \leq n\}$$

a uniform mesh with $x_{i,j} = (ih, jh)$ and $\Delta x = \Delta y = h = 1/n$; see fig. 2.1 for a corresponding representation.

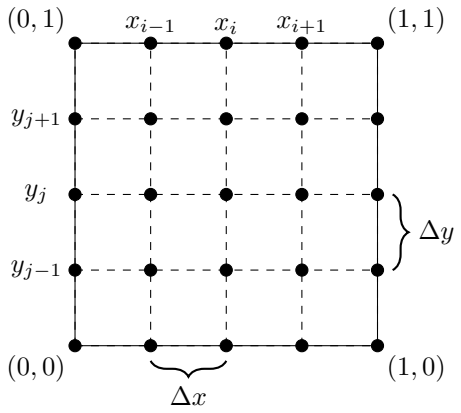


Figure 2.1: A discretization of the unit square with a uniform mesh.

2.2.1 Finite Difference Method

The finite difference method is a class of methods used for solving Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs). In general, this method

consists of discretizing the computational domain by a mesh with a finite set of points and approximating all differential operators present in the equations of interest with finite differences. This converts the differential equations into a finite set of equations, which we can then solve using, for example, an iterative method. Since finite differences are an integral part of this method and we use them multiple times throughout the rest of this work; we discuss them in detail below. Our approach in this subsection is based on [101, 195]. A detailed discussion can also be found in [123].

Let $f : [0,1] \times [0,1] \to \mathbb{R}$ be a smooth function. This means that the function is infinitely continuously differentiable; i.e., its derivatives of all orders exist and are continuous. We are now looking for a way to replace the derivatives of f at a point $x_{i,j}$ by a difference quotient. A suitable tool for this is the Taylor expansion. We step through the procedure of deriving some difference quotients on the example of the first partial derivative f_x of f as an example. Since f is a two-dimensional function we would have to consider multi-dimensional Taylor expansions. For reasons of readability, however, we restrict ourselves to one-dimensional Taylor expansions. Therefore, we hold j constant and omit it from the following consideration. That is, in the following we denote $x_i := x_{i,j}$, $x_i + h := x_{i+1,j}$ and $f' := f_x$. Then, the Taylor expansion of $f(x_i + h) = f(x_{i+1}) = f_{i+1}$ around the point x_i is given by

$$f_{i+1} = \sum_{n=0}^{\infty} \frac{f_i^{(n)}}{n!} h^n = f_i + f_i' h + \frac{f_i''}{2} h^2 + O(h^3), \tag{2.4}$$

where $O(h^3)$ is the big O notation and stands in short for terms of order h^3. Dividing by h and rearranging eq. (2.4) for f_i' gives us

$$f_i' = \frac{f_{i+1} - f_i}{h} - \frac{f_i''}{2} h + O(h^2). \tag{2.5}$$

This representation of f_i' is exact if the number of terms included in the series is infinite or if $h \to 0$. However, we are interested in a *finite* representation. If we truncate this series by dropping all terms containing a derivative of f; i.e., all terms except the first

quotient, we obtain a finite difference representation for the first derivative of f

$$f_i' \approx \frac{f_{i+1} - f_i}{h}. \tag{2.6}$$

By doing so we have introduced an error that is denoted as truncation error. Notice that the lowest order term of the truncation error in eq. (2.5) involves h^1, which is why the finite difference approximation eq. (2.6) is called first-order accurate. We can now also write more accurately

$$f_i' = \frac{f_{i+1} - f_i}{h} + O(h). \tag{2.7}$$

We can further notice that the finite difference representation in eq. (2.6) uses only information to the right of the original grid node x_i. For this reason, this finite difference representation is called forward difference. Altogether, we call this representation first-order forward difference.

In the same manner as before, we can also expand $f(x_i - h) = f_{i-1}$ around the point x_i and obtain

$$f_i' = \frac{f_i - f_{i-1}}{h} + \frac{f_i''}{2}h + O(h^2)\ . \tag{2.8}$$

Using the same procedure as above, we thus obtain the first-order backward difference representation

$$f_i' \approx \frac{f_i - f_{i-1}}{h}. \tag{2.9}$$

We can also combine different Taylor series expansions to obtain higher order approximations. For example, by adding the two expansions eq. (2.5) and eq. (2.8) we obtain

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2)\ . \tag{2.10}$$

Thus by dropping the truncation error we obtain a centered difference representation of f' that is second order accurate

$$f_i' \approx \frac{f_{i+1} - f_{i-1}}{2h}\ . \tag{2.11}$$

In a similar way it is possible to determine finite difference approximations for any other derivative, also of higher order. The grid points on which a finite difference approximation is based are usually called stencil points and the finite difference approximation is consequently called stencil. It is possible to derive an arbitrary stencil based on N grid points for a derivative of order $d < N$. This stencil is then also unique. For a detailed overview of possible finite difference approximations, also on non-uniform grids, we refer to [41]. In the remainder of this work, we denote by $D_h^{k,x} f$ and $D_h^{k,y} f$ the central finite difference approximation of the kth partial derivative of f with regards to the first and second coordinates, respectively, based on a minimum number of stencil points N. For the sake of readability, we will denote the central finite difference approximation of the 1st and 2nd partial derivatives by $D_h^x f$ and $D_h^{xx} f$, respectively, as an exception to this rule.

As a simple example of using the finite difference method to solve a PDE, we consider a stationary diffusion problem on the unit square. That is, we are looking for a function u such that

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= g \quad \text{in } \Omega = [0,1]^2, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{2.12}$$

where g is some suitable function.

For that purpose, we discretize Ω as before with a uniform grid Ω_h. With the finite difference method, however, we do not actually find the solution u but rather an approximation that is defined on the grid nodes. In fact we are now looking for a vector $u^h \in \mathbb{R}^{(n+1)^2}$, where the entries $u_{i,j}^h$ are approximations of the solution $u(x_{i,j})$. This vector is referred to as a grid function. Consequently, we denote by g^h the vector containing the values of g evaluated at the grid nodes $x_{i,j}$. Let us note that the correct notation for a vector should consist of only one index, e.g., u_i^h for the ith entry. In fact, we consider u^h as a vector; however, for simplicity, we use two indices for notation. A single index complicates readability, as can be seen in the relation $u_k^h = u_{i,j}^h$ with $k = (n+1) \cdot i + (j+1)$.

If we discretize the partial derivatives in the equation with the centered differences D_h^{xx}

and D_h^{yy}, we arrive at a system of linear equations

$$\frac{u_{i-1,j}^h - 2u_{i,j}^h + u_{i+1,j}^h}{h^2} + \frac{u_{i,j-1}^h - 2u_{i,j}^h + u_{i,j+1}^h}{h^2} = g_{i,j}^h \quad \forall 0 < i,j < n. \tag{2.13}$$

Due to the boundary conditions we have $u_{i,j}^h = 0$ for $i,j \in \{0,n\}$. This results in a sparse system of linear equations

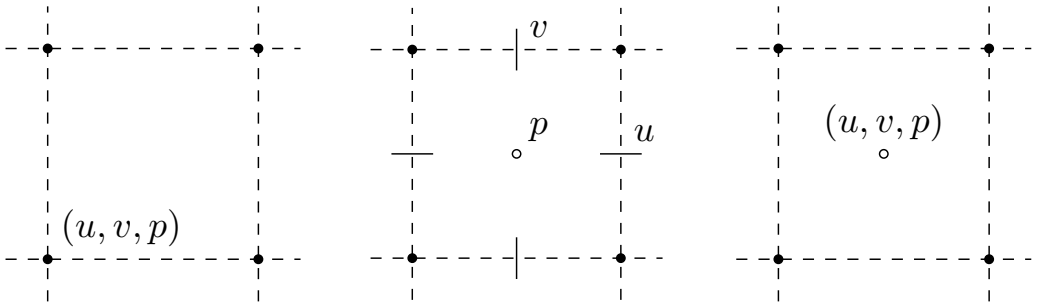
$$Au^h = g^h. \tag{2.14}$$

This system can now be solved by a direct or iterative method; see, for example, [124, 158].

2.2.2 A Finite Difference Discretization of the Incompressible Navier–Stokes Equations

In this section, we will not give a complete derivation and description of a finite difference method for the incompressible Navier–Stokes equations, since the description and derivation of such an algorithm requires a lot of preparatory work, and we will not use such an algorithm in this work anyway. Instead, we present a FD-discretization of the incompressible Navier–Stokes equations and discuss some specifics and problems of the finite difference method with respect to the incompressible Navier–Stokes equations. In eq. (2.1)-(2.2) of section 2.1, we have already introduced the Navier–Stokes equations. First, we discuss the predominant grid configurations used to numerically solve the incompressible Navier–Stokes equations. Afterwards, we turn to discretizing the equations on the uniform grid Ω_h. Then we briefly discuss how the resulting system of equations can be solved. The course of action in this section is oriented on [123, Sect. 2.2.2]. For further literature we also refer to [3].

In general, there are three main types of grid structures that are used to solve the Navier–Stokes equations numerically. Namely, these are unstaggered, staggered and partially staggered; cf. [123]. In fig. 2.2 we show the three different configurations on the example of a single grid cell and indicate the places where the respective variables are calculated. In an unstaggered grid all variables are defined at the grid nodes, also sometimes referred



(a) Unstaggered (b) Staggered (c) Cell-centered, unstaggered

Figure 2.2: The three primarily employed grid structures used for numerically solving the incompressible Navier–Stokes equations.

to as their natural position. Consequently, this grid configuration is also called natural grid; cf. [151]. This arrangement makes it easier for us to implement boundary conditions explicitly without using approximations. In a staggered grid, on the other hand, the variables are all defined in different locations; that is, they are staggered about the grid cell. This configuration was first introduced in [61] in the course of the introduction of the marker-and-cell (MAC) method. On this configuration, the problems associated with pressure-velocity decoupling, cf. section 2.2.3, do not occur. In a cell-centered unstaggered grid all variables are defined at the grid cell centers. This configuration is also called co-located grid. Although it appears to be very similar to the unstaggered grid, there are differences in the approaches that employ this grid configuration; e.g., the variables are stored at the cell centers, but the fluxes are approximated at the cell walls. For more details concerning this grid, we refer to [151]. There are also other grid arrangements that we did not mention, such as the arbitrary Lagrangian–Eulerian grid, introduced in [70]. In the following we consider an unstaggered grid configuration.

As a first step we expand the Navier–Stokes equations in terms of the individual

components

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} - \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0 \quad \text{in } \Omega \tag{2.15}$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} - \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) = 0 \quad \text{in } \Omega \tag{2.16}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \quad \text{in } \Omega, \tag{2.17}$$

where u and v are the x- and y- components of the flow field $\vec{u}$. Second, we consider the grid functions u_h, v_h, and p_h to be approximations of u, v, and p at the grid nodes; that is, $(u_h)_{i,j} = u^h_{i,j} \approx u(x_{i,j})$. Note that for ease of readability, we again used two indices for notation, although u^h, v^h and p^h are vectors, corresponding to our procedure in section 2.2.1. In a third step, we discretize the partial derivatives in the equations 2.15 - 2.17 with centered differences of second order for which we introduce the following notation

$$D^x_h u^h_{i,j} = \frac{u^h_{i+1,j} - u^h_{i-1,j}}{2h}, \qquad D^y_h u^h_{i,j} = \frac{u^h_{i,j+1} - u^h_{i,j-1}}{2h}, \tag{2.18}$$

$$D^{xx}_h u^h_{i,j} = \frac{u^h_{i+1,j} - 2u^h_{i,j} + u^h_{i-1,j}}{h^2}, \qquad D^{yy}_h u^h_{i,j} = \frac{u^h_{i,j+1} - 2u^h_{i,j} + u^h_{i,j-1}}{h^2}, \tag{2.19}$$

and arrive at a set of equations

$$u^h_{i,j} D^x_h u^h_{i,j} + v^h_{i,j} D^y_h u_{i,j} + D^x_h p^h_{i,j} - \nu\left(D^{xx}_h u^h_{i,j} + D^{yy}_h u^h_{i,j}\right) = 0, \tag{2.20}$$

$$u^h_{i,j} D^x_h v^h_{i,j} + v^h_{i,j} D^y_h v^h_{i,j} + D^y_h p^h_{i,j} - \nu\left(D^{xx}_h v^h_{i,j} + D^{yy}_h v^h_{i,j}\right) = 0, \tag{2.21}$$

$$D^x_h u^h_{i,j} + D^y_h v^h_{i,j} = 0, \tag{2.22}$$

for all inner grid nodes $0 < i, j < n$. We can express this set of equations in vector terms

$$u_h \circ D^x_h u_h + v_h \circ D^y_h u_h + D^x_h p_h - \nu\left(D^{xx}_h u_h + D^{yy}_h u_h\right) = 0,$$

$$u_h \circ D^x_h v_h + v_h \circ D^y_h v_h + D^y_h p_h - \nu\left(D^{xx}_h v_h + D^{yy}_h v_h\right) = 0,$$

$$D^x_h u_h + D^y_h v_h = 0,$$

where $\circ$ is the Hadamard product, that is, the element-wise product. Note that we omit the treatment of boundary conditions – and refer for example to [123] for instructions on how to implement them.

We can then reformulate them as a nonlinear system of equations

$$N(\vec{u}_h) + G(p_h) = 0 \quad \text{in } \Omega, \tag{2.23}$$

$$D(\vec{u}_h) = 0 \quad \text{in } \Omega, \tag{2.24}$$

with a nonlinear operator N, linear operators D and G. Here, we consider $\vec{u}_h = \left(u_h^T, v_h^T\right)^T$ to be the discrete velocity vector field. This system, coupled with suitable boundary conditions, can then be solved for the velocity vector and the pressure using Newton's method or Picard iterations; cf. [8]. The existence of a steady-state solution, however, depends on the choice of said boundary conditions and the value of the viscosity ν.

There are many other methods and approaches to solve the incompressible Navier–Stokes equations numerically. Many of them involve deriving a Pressure Poisson Equation (PPE), which is obtained by calculating the divergence of the momentum equation 2.1. In short, the PPE is given by

$$\Delta p = -\nabla \cdot ((\vec{u} \cdot \nabla)\vec{u}) \tag{2.25}$$

The PPE is then used to compute the pressure, while the momentum equations are used to compute the velocity. However, the use of the PPE further complicates the complexity of the required boundary conditions, since additional boundary conditions are now necessary; cf. [53]. Special care must also be given to the discretization of the PPE. We refer to [123] and [40, Chapt. 7] for more details. Other methods solve the unsteady Navier–Stokes equations until they reach a steady state to compute a solution of the steady Navier–Stokes equations; cf. [40]. An example of this approach is the widely used Semi-Implicit Method for Pressure Linked Equations (SIMPLE) algorithm [23, 138][187, Sect. 6.4]. There are of course other methods, among them the previously mentioned

MAC method [61]. We refrain from going into more detail at this point and instead refer to the extended literature [3, 13, 40, 92, 141, 195, 187].

2.2.3 Issues with FDM and Incompressible Navier–Stokes equations

In this section, we present two specific problematic aspects of the incompressible Navier–Stokes equations and possible treatments of them. These are pressure-velocity coupling and the cell-Reynolds problem. This section is based on [123, Sect. 2.2-2.3].

Velocity-Pressure Coupling The choice of an unstaggered grid configuration unfortunately introduces some problems. When discretizing the governing equations as outlined in section 2.2.2 using centered finite-difference approximations, it is possible for non-physical velocity fields to satisfy the discretized divergence-free condition. Similarly, centered approximations of the pressure gradient in the momentum equation enable non-physical pressure fields to remain undetected, and thus uncorrected.

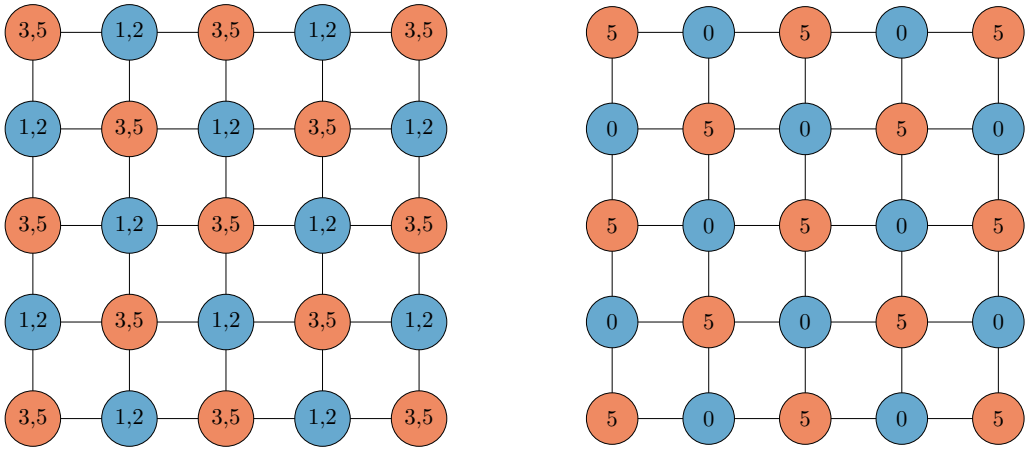


Figure 2.3: Non-physical grid functions satisfying discrete 2D continuity equations. Example of a checkerboard velocity field $\vec{u} = (u, v)$ (left) and a checkerboard pressure field p (right).

To make this problem tangible we provide a simple illustration. The discretized divergence free condition has previously been given in eq. (2.22). If we expand this and assume $h_x = h_y = h$, we obtain

$$u_{i+1,j} - u_{i-1,j} + v_{i,j+1} - v_{i,j-1} = 0. \tag{2.26}$$

Now consider the left of the two grid functions plotted in fig. 2.3, which represents a velocity field that exactly satisfies the discretized equation eq. (2.26), but is physically nonsensical.

The grid function on the right represents a pressure field that remains undetected by centered approximations of the pressure gradients once it occurs. The centered discretizations of the pressure gradients $D_h^x p_{i,j}$ and $D_h^y p_{i,j}$ in the momentum equations eq. (2.20) and eq. (2.21) evaluate to exactly zero on this kind of checkerboard oscillation. Consequently, this pressure field can no longer correct the velocity field via the momentum equations.

In order for the oscillations in the fields in fig. 2.3 to occur, we would have to set non-physical boundary conditions. Since we do not do this, we will not observe oscillations in this pure form in practice. However, these effects do occur in calculations in certain areas of the computational domain when central differences are used; cf. [123].

Another problem that also leads to pressure oscillations is related to the Ladyzshenskaya–Babuska–Brezzi (LBB) condition, also known as the div-stability or inf-sup condition. In short, this condition defines criteria that a discretization must satisfy to be stable, i.e., so that the solution of the discretized system converges to the real solution with finer meshing. This condition is extensively studied in the field of finite element methods; see, e.g., [13] or [141, Sect. 7.2] and references therein. Failure to meet the condition can lead to oscillations in the solution. A review of the condition is technically demanding and exceeds the scope of this section. Therefore, we restrict ourselves to some general statements from the literature with regards to a finite difference discretization of the Navier–Stokes equations. In general, centered finite difference approximations in combination with an

unstaggered grid do not satisfy the inf-sup condition; cf. [123, pp. 68–69]. However, from investigations of the finite element method, for example see [141, Sect. 7.2], we know that it is necessary either to use polynomial approximations of different order for velocity and pressure, mixed finite elements, or, when using the same order, to use different meshes to satisfy the inf-sup condition. In the context of a finite difference approximation, the use of mixed finite elements can be interpreted as the use of a staggered grid; cf. [95, Sect. 3.2].

However, there are ways to get around satisfying the inf-sup condition, i.e., to be able to approximate the velocity and pressure with the same order on the same grid. One of these ways is to introduce a small stabilization term into the divergence-free equation; see [12, Sect. 2.1] or [95, Sect. 4.1]. The continuity equation then takes the form

$$\nabla \cdot \vec{u} = \epsilon \Delta p, \tag{2.27}$$

where ϵ is a (small) constant that can be chosen. Other stabilization terms are also possible, for example $-\epsilon p$. By introducing any of the aforementioned stabilization terms we now have the problem to choose a value for ϵ. If ϵ is too large, the solution will be too distorted, since it is far from divergence-free; if ϵ is too small, oscillations in the pressure can again occur.

Cell-Re Problem and Upwind Schemes The cell-Reynolds (cell-Re) problem is a consequence of certain properties of solutions of difference equations, that is, PDEs approximated with finite differences. Explicitly, it is not a problem arising from the nonlinearity of the NS-equation. It rather stems from certain properties of finite difference approximations of advection-diffusion equations. One possible, if rather heuristic, solution is utilizing upwind approximations of convective terms while another is using a finer resolution. A more detailed mathematical discussion and derivation of the cell-Re restriction for centered differences is given in [123, Sect. 2.3.1]. Here we discuss only the most important points.

The cell-Re problem manifests itself in non-physical oscillations in the solution when

using certain finite difference approximations. This is due to specific properties of finite difference approximations, such as consistency, boundedness, transportiveness, and accuracy. Boundedness, in particular, requires all coefficients of a discretized equation to have the same sign, usually positive; see, for example, [123]. For advection-diffusion equations, centered differences violate this condition for certain values of the so-called cell-Reynolds number. For a consideration of this problem with reference to the finite volume method, we refer to [187, Chapt. 5]. For a similar consideration with regards to the finite element method, we refer to [141, Sect. 8.2]. The analyses performed there, nonetheless, are applicable here as well, as many aspects are similar. For the remainder of this section, however, we follow the procedure in [123, Sect. 2.3.1] and derive the constraint on the cell-Re number via the theory of difference equations.

We consider the Burgers' equation as a 1D model of the incompressible momentum equations of the Navier–Stokes system. Since the nonlinearity and the time dependence are not the cause of the cell-Re problem, we consider here the linearized and steady state form

$$U u_x - \nu u_{xx} = 0, \tag{2.28}$$

where U is a constant. Discretizing eq. (2.28) with centered differences yields

$$\frac{U}{2h}\left(u_{i+1} - u_{i-1}\right) - \frac{\nu}{h^2}\left(u_{i-1} - 2u_i + u_{i+1}\right) = 0. \tag{2.29}$$

If we multiply this by $\frac{h}{U}$, we obtain

$$\frac{1}{2}\left(u_{i+1} - u_{i-1}\right) - \frac{\nu}{Uh}\left(u_{i-1} - 2u_i + u_{i+1}\right) = 0. \tag{2.30}$$

At this point we can then define the cell-Reynolds-number as

$$Re_h = \frac{Uh}{\nu}. \tag{2.31}$$

It can be shown that the solutions of eq. (2.30) are of the form

$$y = c_1 z_+^i + c_2 z_-^i, \quad i = 1, \ldots, N, \tag{2.32}$$

where c_1 and c_2 are constants that depend on initial or boundary conditions and

$$z_+ = \frac{1 + \frac{1}{2}Re_h}{1 - \frac{1}{2}Re_h}, \quad \text{and} \quad z_- = 1. \tag{2.33}$$

If z_+ becomes negative, it is possible that so-called cell-Re wiggles appear in the solution. Here, z_+ actually becomes negative exactly when $|Re_h| > 2$.
We could also, using the definition of Re_h, rewrite eq. (2.30) and obtain

$$\left(-\left(1 + \frac{Re_h}{2}\right) u_{i-1}\right) + 2u_i - \left(1 - \frac{Re_h}{2}\right) u_{i+1} = 0. \tag{2.34}$$

Here, we can immediately see, that $|Re_h| > 2$ leads to at least two coefficients having different signs, which violates one of the boundedness conditions, cf. [187, Sect. 5.4.2]. For this we refer again to [123]; and more generally with regards to the theory of difference equations, we refer to [126].

At this point it should be noted that $|Re_h| > 2$ does not necessarily mean that strong oscillations have to occur in the solution, but only that they can potentially occur. In addition, we have derived the cell-Re restriction using a 1D model equation. Multi-dimensional flow fields pose a more intricate challenge as the solution is influenced by all velocity components and their respective boundary conditions, which can result in local cancellations; cf. [123]. Furthermore, in such scenarios, each grid point has a distinct value for every advective term of the momentum equations instead of a singular value of Re_h.

A common solution to the cell-Re problem is to use upwind or upwind-biased schemes for the convective terms. These are differencing schemes that contain more points on the upwind side. There are many different variants of upwind schemes, the most common being the first-order, the hybrid and the second-order upwind schemes. We refer to the extensive literature for detailed descriptions of the individual schemes, e.g., [123, 138] or [187, Chapt. 5]. However, they usually suffer from other drawbacks, such as numerical diffusion or a lower general accuracy.

2.3 Model Problems

In this section, we describe the Boundary Value Problems (BVPs) that we use throughout this thesis. Here we give descriptions of the geometries, parameters, and boundary conditions that, coupled with the stationary incompressible Navier–Stokes equations, cf. section 2.1, define the BVPs. We describe geometry types rather than individual specific geometries, since we are interested in training surrogate models for different geometries. Therefore, we define the framework in which the geometries of interest are circumscribed.

We consider two different types of geometries. The first type are rectangular channels from which we have cut a star-shaped obstacle. This design is inspired by [35, 58]. We consider this type of geometry in both two and three dimensions, see section 2.3.1 and section 2.3.2, respectively. The second type are non-rectangular geometries that mimic intracranial arteries with aneurysms, see section 2.3.3. Here we consider two subtypes of artery geometries. The first is a bifurcation of an artery, the second is a single artery with a bend. We consider the first artery type in two dimensions only, cf. section 2.3.3.1, and the second artery type in two and three dimensions, cf. section 2.3.3.2 and section 2.3.3.3, respectively. Unless otherwise noted, all lengths in this section are in meters.

2.3.1 Two-Dimensional Flow in a Channel Around an Obstacle

The geometry for the first model problem we consider is a rectangular channel $\Omega = [0, 6] \times [0, 3]$ from which we have cut a star-shaped obstacle, see fig. 2.4. The obstacle is created by randomly choosing a center in the channel and randomly distributing n_P vertices around that center. The location of this center and vertices is constrained so that the obstacle is at least 0.75 away from all boundaries. Additionally, there are constraints to ensure that no acute angles are present, as a sharp-edged obstacle can cause significant problems when constructing an image representation, cf. section 4.4. The randomness of the obstacle placement results in highly variable flow patterns, making this setup a challenging benchmark for a CFD surrogate model.

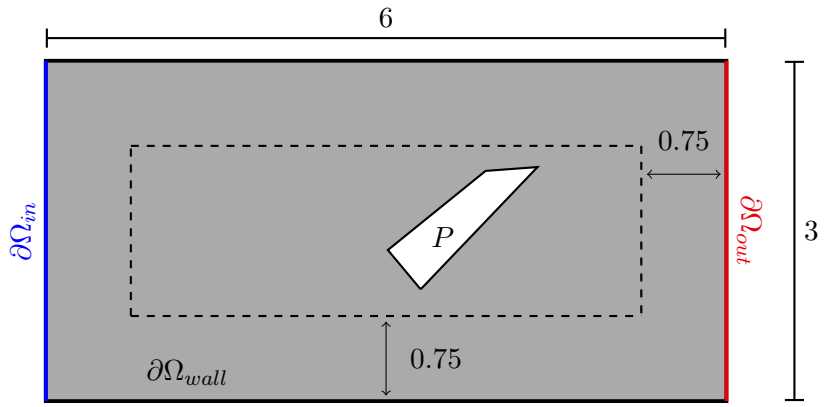


Figure 2.4: Example of a two-dimensional channel geometry Ω with a star-shaped obstacle. The obstacle is confined within a box (dashed line) with a distance of 0.75 to the boundary of Ω.

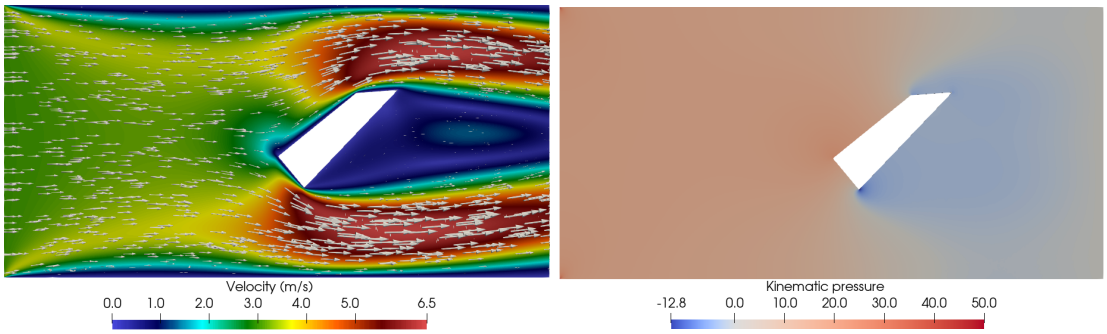
We impose the following boundary conditions: At the inlet boundary $\partial\Omega_{\text{in}} := 0 \times [0, 3]$, we impose a constant inflow velocity $u = (3, 0)^T$, and at the outlet $\partial\Omega_{\text{out}} := 6 \times [0, 3]$, we set the pressure to $p = 0$. The remaining parts of the boundary Ω_{wall}, i.e. the lower and upper boundary $[0, 6] \times 0$ and $[0, 6] \times 3$ as well as the boundary of the obstacle ∂P, correspond to walls, so we enforce no-slip conditions $u = (0, 0)^T$. Finally, we choose $\nu = 5 \cdot 10^{-2}$.

In a channel without obstacles, the Reynolds number is given by

$$Re = \frac{\bar{u}D}{\nu},$$

where $\bar{u}$ is the mean velocity and D is the height of the channel. So in this case we have $Re = \frac{3*3}{5 \cdot 10^{-2}} = 180$.

The result of a sample FV simulation for the obstacle shown in fig. 2.4 is shown in fig. 2.5. For more information about the simulations we use to generate training data in general, see section 4.4.2.



(a) Velocity (b) Pressure

Figure 2.5: Velocity (a) and pressure (b) for a two-dimensional channel geometry obtained from an OpenFOAM simulation.

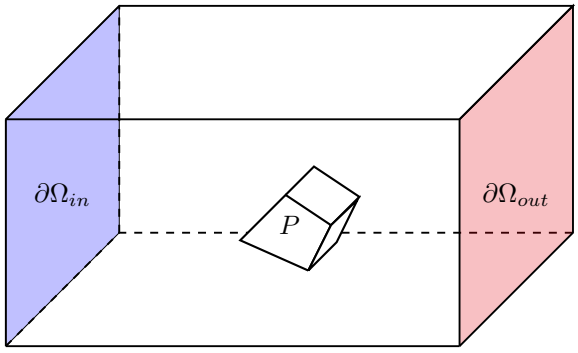
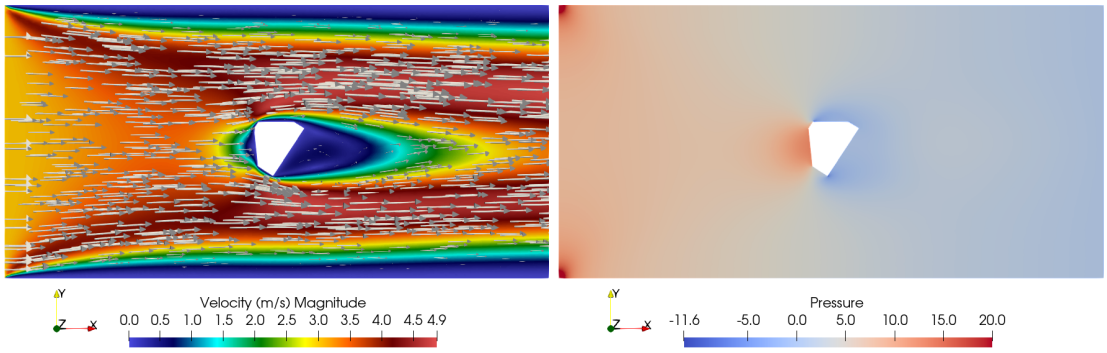


Figure 2.6: Example of a three-dimensional channel geometry Ω with an irregular hexahedron obstacle. The obstacle is confined within a box (not shown) with a distance of 0.75 to the boundary of Ω.

2.3.2 Three-Dimensional Flow in a Channel Around an Obstacle

The basic geometry of the rectangular channel can easily be extended to three dimensions. We simply extrude the two-dimensional channel along a new axis so that the computational domain becomes $\Omega = [0, 6] \times [0, 3] \times [0, 3]$. Accordingly, $\partial\Omega_{\text{in}}$ becomes $0 \times [0, 3] \times [0, 3]$, $\partial\Omega_{\text{out}}$ becomes $6 \times [0, 3] \times [0, 3]$, and Ω_{wall} additionally includes the front and back walls,

i.e., $[0,6] \times [0,3] \times 0$ and $[0,6] \times [0,3] \times 3$, respectively. Since the velocity vector in three dimensions is also three-dimensional, the constant inflow velocity now is given as $u = (3,0,0)^T$ and the no-slip condition is $u = (0,0,0)^T$.



(a) Velocity (b) Pressure

Figure 2.7: Velocity (a) and pressure (b) for a three-dimensional channel geometry, shown here on a slice along the x-axis through the 3D geometry, obtained from an OpenFOAM simulation.

The construction of a three-dimensional obstacle turns out to be more complicated than in the two-dimensional case, since the construction of an arbitrary polyhedron is more difficult than the construction of an arbitrary polygon. Therefore, we restrict ourselves to polyhedra with 4 to 11 vertices, which by construction follow some order, but are still sufficiently random and irregular. Thus, we create an obstacle with 4 vertices as an irregular tetrahedron. We create an obstacle with 5 to 7 vertices by starting from an arbitrary tetrahedron, then iteratively adding a vertex at random and merging it with the nearest face of the previous polyhedron to form a tetrahedron, and finally merging the old polyhedron with the new tetrahedron to form a new polyhedron. We create an obstacle with 8 vertices as an irregular hexahedron with quadrilaterals as faces. Obstacles with 9, 10 or 11 vertices are created in the same way as obstacles with 5 to 7 vertices, i.e. by extending the basic polyhedron with tetrahedra, but this time starting from a hexahedron.

This allows us to create a wide variety of obstacles, which are generally non-convex. We apply the same restrictions to the obstacle as in the two-dimensional case, that is, there are no acute angles, so that the obstacle exhibits no pointy vertices or sharp edges. Furthermore, the obstacle is again at least 0.75 away from all boundaries. See fig. 2.6 for an example three-dimensional channel geometry with a hexahedron obstacle. The result of a sample FV simulation for a three-dimensional channel geometry is shown in fig. 2.7.

2.3.3 Intracranial Arteries with Aneurysms

There are two types of intracranial, or cerebral, aneurysms: saccular and fusiform [202]. A saccular aneurysm is an outward saccular bulge of the artery wall, while a fusiform aneurysm is a ball like expansion of the whole artery. Saccular aneurysms are more prevalent than fusiform aneurysms [202]. They are most commonly located at bifurcations of the Circle of Willis [198]. In [76], ruptured intracranial aneurysms were reported to be most commonly located on the anterior communicating artery (ACoA), middle cerebral artery (MCA), and posterior communicating artery (PCoA). See also [5] for three examples of intracranial saccular aneurysms.

We consider two two-dimensional types of geometry: a bifurcation with a saccular aneurysm in close proximity to the bifurcation, see section 2.3.3.1; and a single curved artery, also with a saccular aneurysm on the inside of the bend, see section 2.3.3.2. Additionally, we consider one three-dimensional type of geometry, a single curved artery with a fusiform aneurysm on the bend, see section 2.3.3.3. The sizes of the model arteries have been chosen to be similar to those found in the Circle of Willis. For example, the A1 portion of the anterior cerebral artery (ACA) is reported to have a diameter of approximately 2 mm [90] and the MCA is reported to have a diameter of approximately 2.5 mm to 4 mm [47]. Accordingly, our arteries have diameters of 2 mm to 4 mm.

We assume a kinematic viscosity of $4.0e-06\frac{m^2}{s}$ for all considered geometries, which corresponds to that of human adult blood [155]. Unless explicitly stated otherwise, we

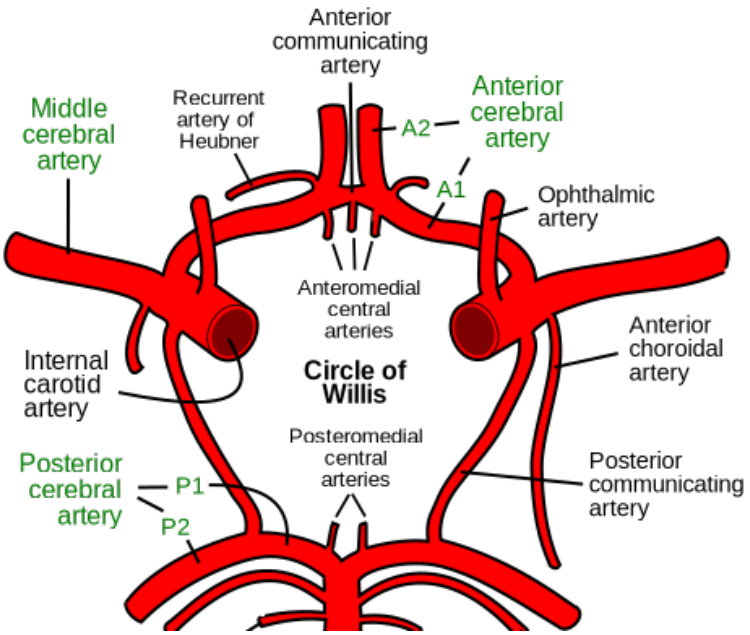


Figure 2.8: Diagram of the circle of Willis. Taken from [150].

also consider a uniform inflow velocity of $0.1\frac{m}{s}$ at the inflow boundary and set p to 0 at the outflow boundaries. At the arterial walls we set a no-slip boundary condition. With the chosen parameters, this general setup resembles a large cerebral artery [192] with normal blood flow at a flow rate of slightly more than $1\frac{ml}{s}$ [192, 205], with smaller flow rates for arteries of smaller diameter.

The Reynolds number in both a two-dimensional and a three-dimensional artery in the absence of an aneurysm is given by

$$Re = \frac{\bar{u}D}{\nu},$$

where $\bar{u}$ is the mean velocity and D is the width of the artery. Thus, we have $Re = \frac{0.1 \cdot 0.004}{4 \cdot 10^{-6}} = 100$. This is consistent with intracranial blood flow, where Reynolds numbers of the order of 10^2 typically occur [5].

2.3.3.1 Two-Dimensional Bifurcation Geometries

The geometry for this model problem is a bifurcation of cerebral arteries of the circle of Willis. The geometry consists of an incoming artery that bifurcates into two outgoing arteries. We keep the incoming artery and the basis of the bifurcation constant and vary the angle and width of the outgoing arteries. There is also a saccular aneurysm near the bifurcation. This aneurysm varies based on several parameters, including neck width, conicity, aspect ratio, position at the bifurcation, and rotation relative to the bifurcation. The meaning and scope of these parameters will be explained later. We constrain the domain of interest by a square bounding box that is 4 *cm* wide and 4 *cm* high. In fig. 2.9 we show a diagram of this geometry.

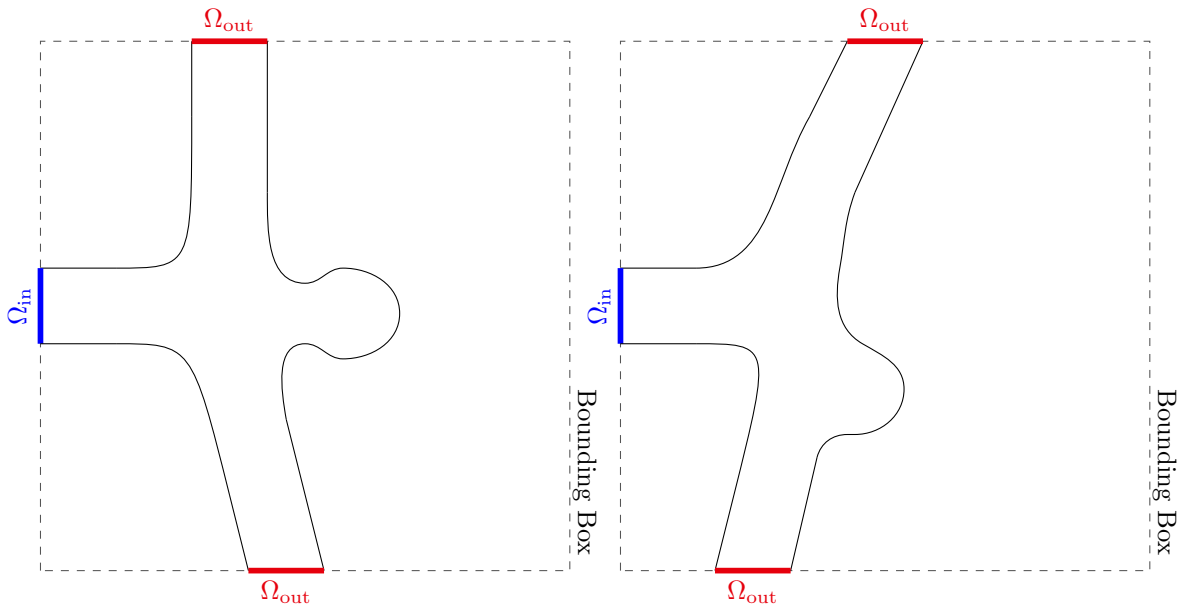
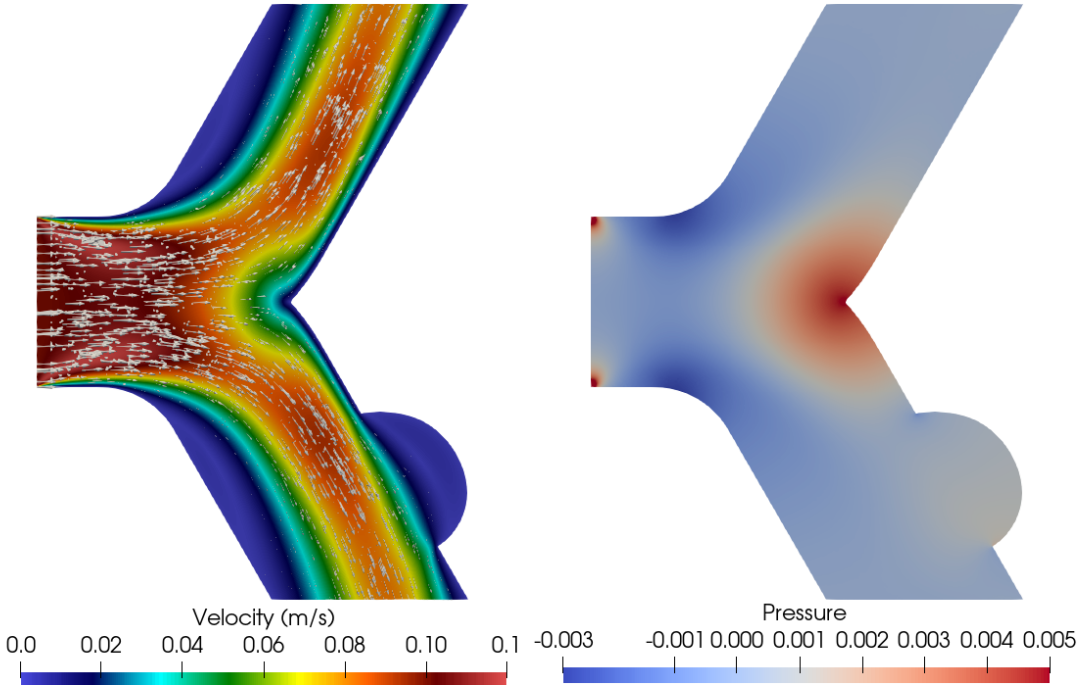


Figure 2.9: Examples of a two-dimensional geometry of a bifurcation Ω with an aneurysm.

The incoming artery has a uniform width of 0.004 *m* or 4 *mm* at the inflow. The

incoming artery divides into two arteries, each at an initial angle of 90° to the incoming artery. The outgoing arteries additionally bend at an angle between 30° towards the incoming artery and 75° away from it, relative to their initial orientation, labeled od_1 and od_2. The width of the outflow arteries, labeled ts_{o1} and ts_{o2}, varies between 3 mm and 4 mm.



(a) Velocity (b) Pressure

Figure 2.10: Velocity (a) and pressure (b) for a two-dimensional bifurcation geometry obtained from an OpenFOAM simulation.

The aneurysm is described by a number of parameters that affect its shape, orientation, and location. Characteristics of an aneurysm include its height, diameter, and neck width. The neck width is the size of the entrance of the aneurysm to the artery. The diameter is measured at the widest point of the aneurysm. The height is the distance from the neck to the dome, which is the point opposite the artery in the aneurysm. We control

these values indirectly by defining four parameters that determine them with respect to a reference aneurysm. The reference aneurysm has a height of 4 mm, a diameter of 4 mm, a neck width of 2 mm, and a dome width of 2 mm. The parameters we control are: aspect ratio, defined as height divided by diameter; conicity, a measure of where the diameter is greatest; and bottleneck factor, defined as maximum width divided by neck width. The ranges in which we vary these parameters are $[0.25, 0.75]$ for the aspect ratio, $[-1, 1]$ for the conicity, and $[1, 4]$ for the bottleneck factor. Here, a conicity of -1 indicates that the widest diameter is assumed to be near the neck, and a conicity of 1 indicates that the widest diameter is assumed to be near the dome. There is one additional factor with which we control the scale of the aneurysm in general, that is the relative aneurysm size. This is consistent with the size of aneurysms reported in the literature [76]. The result of a sample FV simulation for a bifurcation geometry is shown in fig. 2.10.

2.3.3.2 Two-Dimensional Single Artery Geometries

Additionally, we consider a second model problem based on a geometry that represents a different type of artery-aneurysm combination. This geometry consists of a single curved artery with a saccular aneurysm on the inside of the bend.

This geometry depends on fewer parameters than the geometry of the bifurcation we discussed in the previous section. There are two parameters that describe the artery, the diameter of the artery and the kink angle. The diameter of the artery varies between 0.002 m and 0.004 m and the bend angle between 0° and 130°. Two other parameters characterize the aneurysm. The aneurysm always has the same shape and we vary the size of the aneurysm and its position along the artery. The size of the aneurysm is determined relatively to a reference aneurysm and varies between 0% and 130%. The reference aneurysm has a height of 4 mm, a diameter of 6 mm, a neck width of 3 mm, and a dome width of 3 mm. The aneurysm is placed along the artery in the region of the bend with a parameter between 0 and 1.

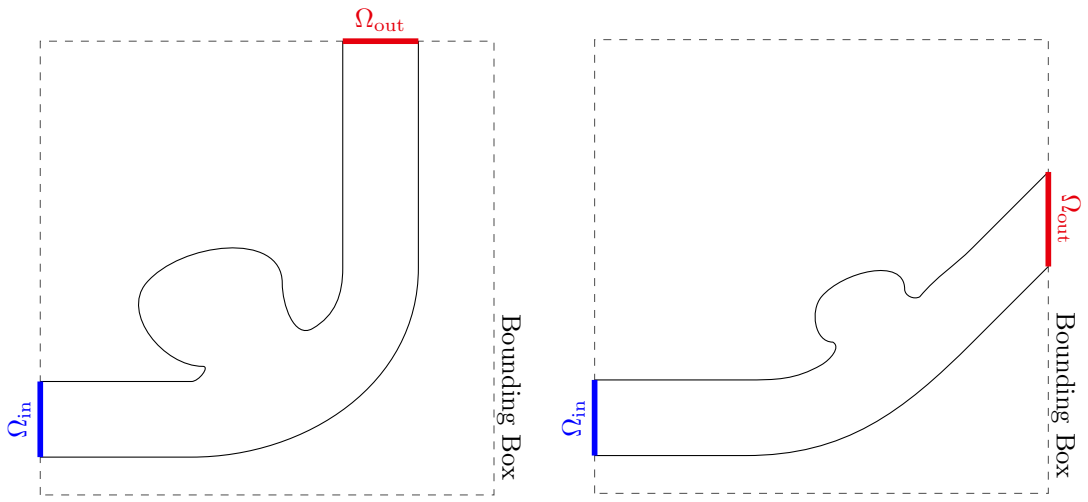


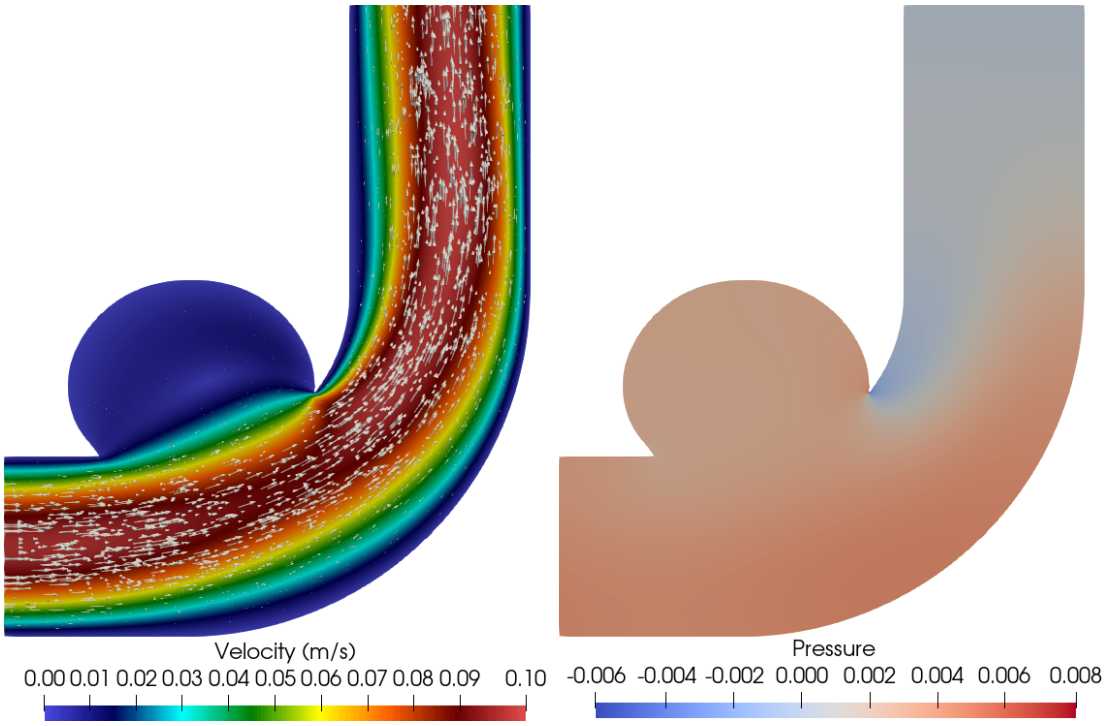
Figure 2.11: Examples of a two-dimensional geometry of an individual artery Ω with an aneurysm.

In this geometry, we do not set a constant inflow velocity at the inflow boundary, but a parabolic one. We choose it so that the flow rate is equal to the constant inflow velocity. The result of a sample FV simulation for a single artery geometry is shown in fig. 2.12.

2.3.3.3 Three-Dimensional Single Artery Geometries

The three-dimensional geometry we are considering is similar to the two-dimensional single artery geometry. It consist of a single curved artery with an aneurysms located along its bend. The main difference is that, here, we have a fusiform aneurysm rather than a saccular aneurysm.

For this model problem, we choose the diameter of the artery to be constant at 4 mm. The geometry depends on three parameters, the width and height of the fusiform aneurysm, and the angle of bend of the artery. The width is measured from the point where the diameter of the artery begins to increase to the point where it returns to its normal



(a) Velocity (b) Pressure

Figure 2.12: Velocity (a) and pressure (b) for a two-dimensional single artery geometry obtained from an OpenFOAM simulation.

diameter. Height is the maximum increase in the radius of the artery. We vary the width from 8 mm to 12 mm and the height from 4 mm to 8 mm. The bend angle of the artery varies from 0° to 130°.

In this geometry, we again set a parabolic inflow velocity, just as in the two-dimensional case. We choose it so that the flow rate is equal to a hypothetical constant inflow velocity. The result of a sample FV simulation for a single artery geometry is shown in fig. 2.14.

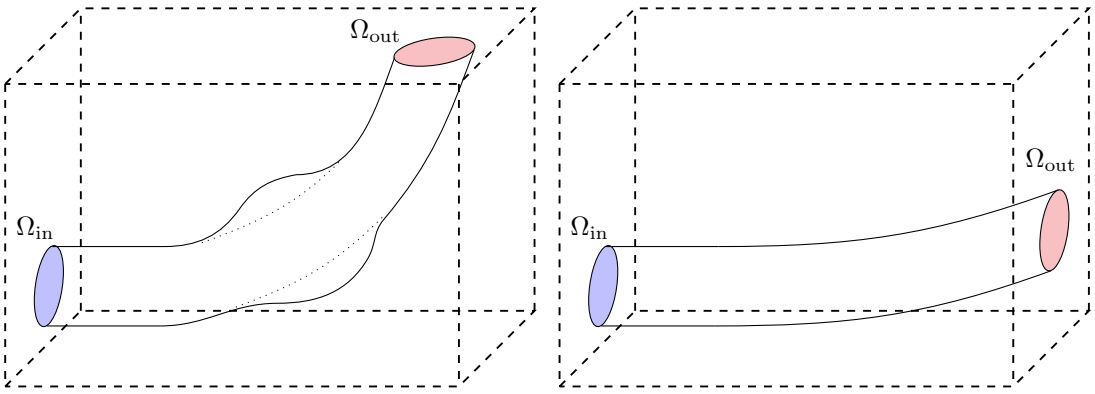
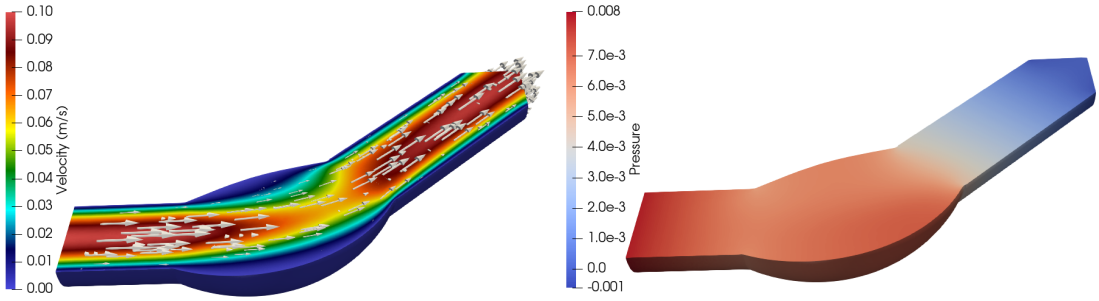


Figure 2.13: Examples of a three-dimensional geometry of an individual artery Ω with an aneurysm.



(a) Velocity (b) Pressure

Figure 2.14: Velocity (a) and pressure (b) for a three-dimensional single-outflow geometry, shown here on a slice along the x-axis through the 3D geometry, obtained from an OpenFOAM simulation.

3 Deep Learning

With the increased availability of data and computational resources to process this data, the field of machine learning has experienced a renaissance in the 2010s, especially the subfield of deep learning. On the broadest level, machine learning models and algorithms are designed to process large data volumes and extract underlying patterns, i.e., learn or gain experience, in order to generalize with respect to a specific task on previously unseen data [129]. This can be a task as simple as forecasting the next number in a given series using the preceding numbers as input, or as complex as forecasting the weather for the next seven days using past meteorological measurements as input. Other common machine learning tasks include image classification [91, 154, 177], speech recognition [54, 69], medical diagnosis [87], credit card fraud detection [4], and data clustering [75].

Deep learning [52] is a subfield of machine learning that comprises of methods based on artificial neural networks. Neural networks are inspired by the structure and function of the human brain and consist of layers of interconnected nodes that process and relay information. Deep learning models have achieved state-of-the-art results in tasks such as image recognition [137], image segmentation [128], and natural language processing [136]. As a further subdivision, machine learning methods, and by extension deep learning methods, are often categorized as supervised, unsupervised, or reinforcement learning. However, it is also possible for a method to be classified as both supervised and unsupervised learning [193]. For a more comprehensive overview of deep learning, see [131, 52, 62, 83].

In the following, we give an overview of two common types of neural networks, namely Dense Neural Networks (DNN) in section 3.2 and Convolutional Neural Networks (CNN) in section 3.3. In section 3.4 we discuss common problems in training machine learning models.

3.1 Machine Learning

As mentioned earlier, machine learning methods learn from experience with a given task and performance measure [129]. In this section, we explain these rather abstract terms using examples appropriate to our application, and lay the groundwork for the following section. For a more detailed overview of machine learning in general; see, for example, [52, Ch. 5][193].

Machine learning methods are used for a wide variety of tasks. Among them are regression and classification. Given that the approach proposed in this thesis is based on a regression task, we will exclusively focus on regression and only remotely mention other tasks. All machine learning methods have in common that they are fundamentally based on data from which they learn or gather experience. For supervised methods the used data broadly consist of input data, often called features, and corresponding output data, often called labels or target. Performance is then a measure of, for example, how close the prediction of the machine learning model is to the target, given the corresponding input. Here, experience is gained from evaluating performance and updating the machine learning model accordingly. In unsupervised methods, the data consists only of input data. The machine learning model is not presented with target data, but instead finds commonalities or structure in the input data. Examples of unsupervised learning include dimension reduction and clustering.

A supervised machine learning model f^{Ψ}, with model parameters Ψ, solving the

regression task approximates a, usually, nonlinear function f^*

$$\begin{aligned} f^* : \mathbb{R}^{n_i} &\to \mathbb{R}^{n_o} \\ f^*(x) &= y, \end{aligned} \tag{3.1}$$

where n_i and n_o are the dimensions of the input and output spaces. In order for the model to learn to predict y from x, it is trained on a set of n_t pairs of input vectors x_i and corresponding output vectors y_i denoted by $T = \{(x_i, y_i)\}_{i=1}^{n_t}$. Commonly, T is called the *training data set*. Usually this training data set is accompanied by a *validation data set* $V = \{(x_i, y_i)\}_{i=1}^{n_v}$, consisting of unseen data pairs, which is used to validate the performance of our model, i.e., how well the model f^Ψ approximates f^*. The validation data set is sometimes also called the *test data set*. Typically, however, the validation and test data sets are different data sets, and the test data set is used to validate the performance of a model after its hyperparameters have been tuned to perform best on the validation data set. Hyperparameters are parameters that describe a model or control the learning process and, unlike the model parameters Ψ, cannot be derived by training. Examples of hyperparameters include the learning rate (or step size) used in gradient-based optimizers, or the shape and size of a neural network.

To evaluate the performance, a cost or loss function l is defined:

$$\begin{aligned} l : \mathbb{R}^{n_o} \times \mathbb{R}^{n_o} &\to \mathbb{R} \\ l(\hat{y}, y) &= s, \end{aligned} \tag{3.2}$$

which assigns a score s to a pair of a prediction $\hat{y}$ and a ground truth y. Since the prediction $\hat{y}$ is the application of the model f^Ψ to the input x, i.e., $f^\Psi(x) = \hat{y}$, we can say that a score is assigned to each data pair (x, y). Thus, the loss could also be defined as a mapping from (x, y) to s. In general, a low value, or loss, is associated with a good fit, while a high loss is associated with a poor fit. The choice of loss function depends on the problem we are trying to solve. Common choices include the sum of squared errors (SSE), the mean squared error (MSE), and the root mean squared error (RMSE).

A supervised machine learning model is typically trained by minimizing eq. (3.2) with respect to the model parameters Ψ. That is, the minimization problem

$$\arg\min_{\Psi} \frac{1}{|T|} \sum_{(x,y)\in T} l\left(f^{\Psi}(x), y\right) \tag{3.3}$$

which we can solve, for example, using gradient descent, its more sophisticated advancements, such as the Adam optimizer [85], or even quasi-Newton methods such as the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm (L-BFGS) [110]. Of course, since the minimization problem is defined over the training data set T, its size and composition play a significant role in the performance of the trained model. If the trained model has been fitted too closely to the training data, weakening its generalization ability, i.e., if the evaluation of the loss function on the validation data set yields significantly higher values than on the training data set, then this is referred to as overfitting [204]. On the other hand, the model should also be able to make accurate predictions for previously unseen data, thus avoiding underfitting [52, Sec. 5.2].

3.2 Dense Neural Networks

Dense Neural Networks (DNNs), also called Feedforward Neural Networks (FNN), Multilayer Perceptrons (MLPs), or Artificial Neural Networks (ANNs), are the most basic deep learning models. Due to the universal approximation capability of neural networks [72], they have been widely used in a variety of problems. As they lay the foundation for more complex and specialized models as well as several models described in chapter 4, we give a mathematical introduction to them; cf. [131, pp.106-121], [52, Chapt. 6], [193, Chapt. 13]. Our approach in chapter 5 is based on CNNs, not DNNs. However, many of the aspects we discuss about DNNs also apply to CNNs, which we introduce in more detail in the next section.

In general, DNNs can be used to approximate a, usually nonlinear, function f^*. Often, this function is either a continuous or a categorical mapping, so that the task of approxi-

mating f^* is a regression or classification task, respectively. For example, let $f^*(x) = y$ be a function that assigns an output $y \in \mathbb{R}^m$ to each input $x \in \mathbb{R}^n$. Then a neural network is a function $f^\Psi(x) = \hat{y}$ that approximates the function f^*. Here, Ψ are the network parameters. More precisely, a DNN consists of a number of consecutive layers of nodes,

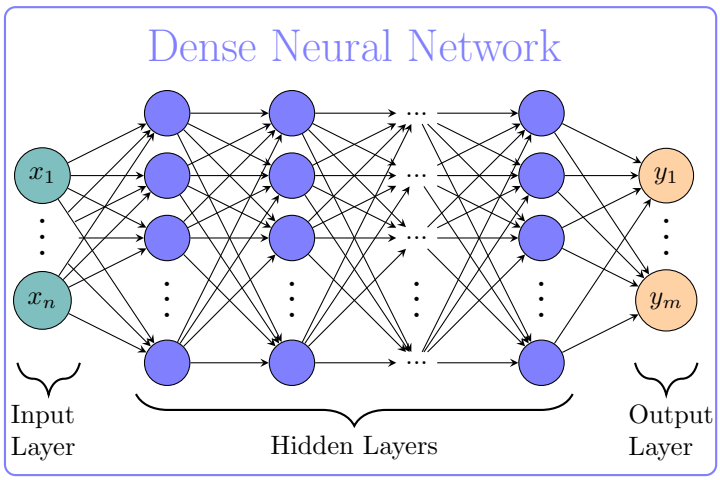


Figure 3.1: Graph representation of the structure of a dense neural network.

also called neurons or units. Conceptually, nodes are based on the design of neurons in the human brain [16]. Each node in a layer is connected to all nodes in the previous layer, which is why such a neural network is called dense. In practice, a connection between two nodes takes the form of a real number that is called a weight. A schematic of a DNN is shown in fig. 3.1, where the weights are represented by the edges between nodes. The first layer is associated with the input x and the last layer with the output y. The layers in between are called hidden layers. In each layer, except for the first layer, a node takes the output of the previous layer, computes a weighted sum, adds a bias, and applies a nonlinear activation function. Thus, the output x^j of the jth layer is given by

$$x^j = f_j^{\Psi_j}(x^{j-1}) = \alpha^j \left(W^j x^{j-1} + b^j \right), \tag{3.4}$$

where x^{j-1} is the output of the previous layer, $W^j \in \mathbb{R}^{n_j \times n_{j-1}}$ is the weight matrix

containing the weights for all connections between the $(j-1)$th and jth layers, $b^j \in \mathbb{R}^{n_j}$ a vector containing the biases for all nodes in the jth layer, and α^j an activation function. Except for the activation function, this computation is linear, and thus it is the nonlinearity of the activation function that gives a neural network the ability to approximate highly complex functions f^*.

The application of a DNN with N hidden layers to an input x is given by

$$\begin{aligned} x^1 =& \alpha^1\left(W^1 x + b^1\right), \\ x^j =& \alpha^j\left(W^j x^{j-1} b^j\right), \quad 1 < j < N, \\ \hat{y} =& W^{N+1} x^N + b^{N+1}. \end{aligned} \tag{3.5}$$

Typically, no activation function is applied to the output of the last layer if the network is used for a regression task, and the sigmoid or the softmax activation function is applied if the network is used for binary or multiclass classification.

For a given training data set $T = \{(x_i, y_i)\}_{i=1}^{n_t}$ and the MSE as the loss function, training this DNN consists of solving the minimization problem

$$\arg\min_{\Psi} \frac{1}{|T|} \sum_{(x,y)\in T} l\left(f^{\Psi}(x) - y\right)^2, \tag{3.6}$$

where Ψ is the set of the weight matrices W^j and bias vectors b^j of the layers of the DNN. Typically this minimization problem is solved using gradient-based methods, such as the gradient descent method or more advanced methods such as the Adam optimizer [85].

In the context of neural networks the gradients of the loss $l(\hat{y}, y)$ with respect to each parameter of the network are computed by the backpropagation algorithm [157], also known as the reverse mode of automatic differentiation (AD) [7]. This algorithm exploits the fact that the application of a DNN, also referred to as forward propagation or forward pass, is a chaining of elementary operations, such as addition and multiplication. By clever use of the chain rule, the partial derivatives of the loss function with respect to each parameter can be determined. To do this, the computational graph of the DNN

is traversed backwards starting from the loss. This process is called backpropagation or backward pass. For more details on backpropagation; see [52, Sect. 6.5] and [193, Appendix B].

There are many choices available for the activation function α and theoretically it is possible to use a different activation function for each individual node. However, in the author's experience, it is common to choose one activation function and use it for all nodes in the network. Most of the available activation functions are simple mathematical functions or combinations of them. For example, the widely used rectified linear activation unit (ReLU) function [50] is defined as

$$\alpha(x) = \max(0, x) .$$

Other common choices include the hyperbolic tangent and the sigmoid function. There are many more possible functions, just to name a few: Gaussian error linear unit (GELU) [66]; sigmoid linear unit (SiLU), also known as swish [145]; and leaky rectified linear unit (Leaky ReLU) [132].

There are many other commonly used techniques in the context of DNNs to improve the performance and generalization properties of the trained models. For example, batch normalization [74] to speed up learning, regularization [52, Chapt. 7] to reduce overfitting, or skip connections [64] to allow training of deeper networks. Depth refers to the number of consecutive computational layers. The above methods are not exclusive to DNNs and can also be used in the context of CNNs, for example.

3.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [97] are a special type of neural network for data that has the structure of a tensor product. This includes time-series (1D) and pixel images (2D), but also voxel images (3D). CNNs use a special linear operator, a convolution, to operate on the image data. They have several advantages over DNNs, e.g., a convolutional

layer requires significantly fewer parameters to process the data than a fully connected dense layer. Convolutions generally take advantage of the structure of the data, which is also their disadvantage, since they depend on this structure. Excellent results have been achieved with CNNs in areas such as image recognition [27, 91, 98] and natural language processing [11, 54, 78, 169]. In general, a CNN is a neural network that uses a convolution in at least one of its layers [52, Chapt. 9]. We will refer to a network as a fully convolutional neural network if it consists only of convolutions and image manipulation methods; cf. [112]. CNNs form the basis of our primary approach presented in chapter 5. In this section, we will give an introduction to convolutional neural networks. We start with a mathematical description of convolutions before moving on to convolutions in the context of neural networks. This section is based on [52, Chapt. 9].

In general, a convolution, denoted by an asterisk, is an operation on two functions $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ defined by

$$(f * g)(x) = \int f(s)g(x - s)ds. \tag{3.7}$$

Here, $f(x)$ could be a function that gives us measurements of something of interest over an area and $g(x)$ could be a weight function, such as a probability density function. In this case $(f * g)$ would be something like a weighted average of f. However, the data we train our models on is not continuous, but discrete, so we consider the discrete convolution

$$(f * g)(x) = \sum_s f(s)g(x - s). \tag{3.8}$$

With convolutional neural networks, f is usually called input, g kernel, and $(f * g)$ the output feature map [52].

In a machine learning context, the data or input we are working with is usually a multidimensional array of data and the kernel is correspondingly a multidimensional array of parameters or weights. For example, if $I \in \mathbb{R}^{w \times h}$ is a two-dimensional input, such as a pixel image, and $K \in \mathbb{R}^{k \times l}$ is a two dimensional kernel, then the two-dimensional discrete

convolution is given by

$$C_{i,j} = (I * K)_{i,j} = \sum_m \sum_n I_{m,n} K_{i-m,j-n}. \tag{3.9}$$

Since a convolution is commutative we can flip the kernel relative to the input and equivalently write

$$C_{i,j} = (I * K)_{i,j} = \sum_m \sum_n I_{i-m,j-n} K_{m,n}. \tag{3.10}$$

In this notation, the size of K determines the range of values of m and n. Since in practice K is usually much smaller than I, the sum is more compact in this form. In general, however, ML frameworks do not implement this form of convolution, but a slightly modified version, called cross-correlation:

$$C_{i,j} = (I * K)_{i,j} = \sum_m \sum_n I_{i+m,j+n} K_{m,n}. \tag{3.11}$$

To stay in the context of machine learning, we will adopt the habit of calling this function convolution. An example of a discrete convolution is shown in fig. 3.2. Note that in this introduction to discrete convolutions, we intentionally overloaded the $*$ operator. By doing so, we avoided having to introduce additional notation that would not be used anywhere else in this thesis.

A CNN uses such a convolution in at least one layer, and we will call such a layer a convolutional layer. In a convolutional layer, there is rarely just one convolution applied, but rather several in parallel. In addition, the input to a convolutional layer is usually not just a grid of scalars, but a grid of vectors. For example, a color image consists of three color values at each pixel. Furthermore, a CNN usually consists of several successive convolutional layers, so that the input of one layer is usually the output of the previous one. Overall, a convolution in a convolutional layer does not operate on a two-dimensional image, but on a three-dimensional tensor with one index for the different channels and two indices for the spatial coordinates of each channel. Normally, there is a fourth dimension that refers to the examples in a batch, but we omit this dimension for ease of reading. As

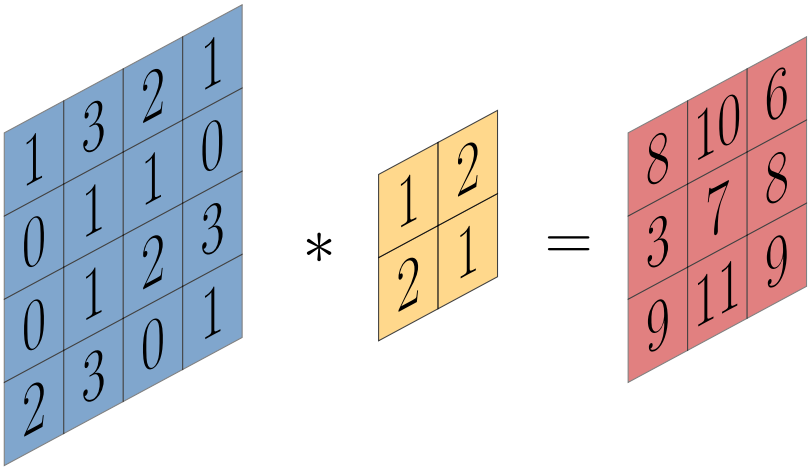


Figure 3.2: Example of a discrete two-dimensional convolution of a 2×2 kernel on a 4×4 input.

in the case of a fully connected layer, the application of a convolution corresponds to a linear sum. Therefore, the use of a nonlinear activation function is also necessary here.

Applying a convolutional layer C with a four-dimensional kernel $(K_{c_{in},c_{out},i,j})$ and an activation function α to an input $(I_{c_{in},i,j})$, where an entry of K defines the relationship between an entry in channel c_{in} of the input and an entry in channel c_{out} of the output with an offset of i rows and j columns between the output entry and the input entry, is then defined by

$$O_{c_{out},i,j} = C(I, K, \alpha) = \alpha \left(\sum_{c,m,n} I_{c,i+m,j+n} K_{c,c_{out},m,n} \right). \tag{3.12}$$

Consequently, as with DNNs, cf. section 3.2, we can recursively define the application of a CNN consisting of N successive convolutional layers to an input I as

$$\begin{aligned} O^1 &= C(I, K^1, \alpha^1), \\ O^l &= C(O^{l-1}, K^l, \alpha^l) \quad 1 < j < N, \\ O^N &= C(O^{N-1}, K^N, id). \end{aligned} \tag{3.13}$$

Here, the activation function of the last layer is the identity. This is useful for regression tasks, since using an activation would limit the output of the neural network to the range of the activation function used. Of course, it is also possible to use CNNs for classification tasks. The same applies as for DNNs. We can also construct a neural network from a combination of convolutional and fully connected layers.

In general, there are additional building blocks that are used in a CNN. These include: pooling layers to reduce spatial and/or feature dimensions [164]; transposed convolutional layers, also called fractional strided convolutional or deconvolutional, to increase spatial dimensions; cf. [203, 206]. There are also many possible variations of convolutions commonly used in CNNs, but we will only cover some of them, and only briefly – namely padding and striding. In general, we refer to [34] for a detailed and graphically illustrative overview of the various operations used in CNNs.

Without padding, the output of a convolutional layer is always smaller than the input in spatial dimensions. This effect would severely degrade the performance of a deep CNN. To avoid this, the input of a convolutional layer is usually padded with zeros at the edges. One extreme is to keep the size of the output image equal to the size of the input image. In TensorFlow, this is called *same* padding. The other extreme of no padding is called *valid* padding.

If we are interested in reducing the spatial dimension of the images, we can use a technique called striding, which means skipping some positions of the image. Here we only work with every s-th pixel, where s is called *stride*. We can use different strides for each dimension. In practice, this is equivalent to downsampling the convolution result. For example, a stride of 2 roughly halves the spatial dimensions.

An example CNN architecture with 6 convolutional layers is shown in fig. 3.3. This model reduces an input of size $N \times N$ in the spatial dimensions by using strided convolutions while hopefully extracting useful features. For example, this could reduce the spatial dimensions to 1×1 while increasing the feature dimension to k. Applying the softmax

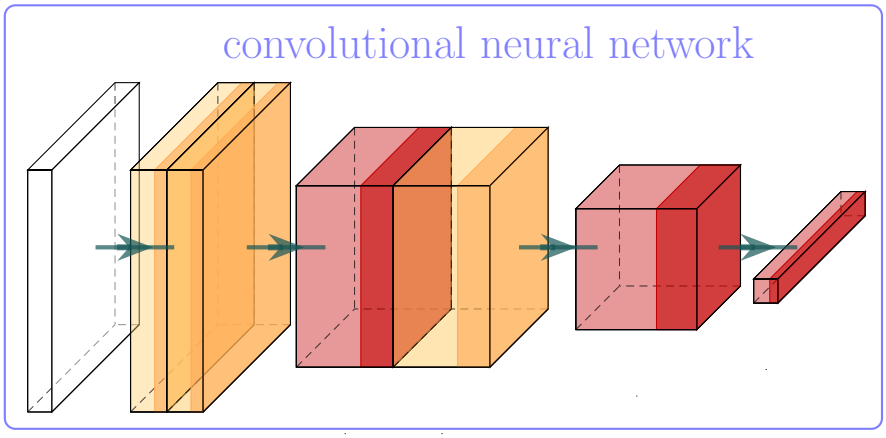


Figure 3.3: Example architecture of a CNN. On the left side is the input in the form of a two-dimensional pixel image. This network consists of two different types of convolutional layers. Both the yellow and red blocks represent convolutional layers with same padding, the yellow one with a stride of 1 and the red one with a stride of 2 or more.

function to the last layer allows us to train this CNN as a classification network for k classes.

Using convolutions has several advantages over a dense layer. For example, the same weights are used for each entry of the output C, unlike a fully connected layer where each neuron in a layer has its own set of weights. This is called *parameter sharing*.

In practice, K is several orders of magnitude smaller than I. As a result, only a few values of the input are used to calculate an output value. The interaction between input and output is therefore sparse, and this property is called *sparse connectivity*. In fact, one can represent a convolution as a fully connected layer where the weight matrix is sparse.

The two previous properties together yield another property, that of *equivariance to translations*. Applying a convolution to a shifted input produces the same output as

applying a convolution to the original input and shifting the output. However, this property does not apply to CNNs in general, since other operations are involved that are not shift-invariant.

3.4 Problems and Pitfalls when Training Neural Networks

There are a number of problems associated with training ML models, and neural networks in particular. In this section we briefly discuss two of the most important ones. However, we do not address overfitting and refer to [204] for an overview of overfitting and possible solutions.

3.4.1 Non-Convexity of Loss function

Previously we mentioned the universal approximation capability of neural networks [72]. However, this property refers only to approximability, not to trainability and generalizability. Training a neural network means minimizing its loss function, a function that almost always is highly nonconvex [193, Sect. 13.5]. In particular, loss functions often have large flat regions, local minima and saddle points [52]. Typically these nonconvexities are problematic for first-order optimization methods, especially gradient-based methods. There are improvements such as momentum-based and normalized gradient methods that are better able to deal with this type of nonconvexity. Of particular note is the Adam optimizer, which determines individual adaptive learning rates for various parameters from estimates of the first and second moments of the gradients; cf. [85]. In comparison to other first-order gradient-based methods, this method has repeatedly been shown to have superior performance; see for example [85, 156].

However, some networks, especially deep neural networks consisting of a large number of layers, exhibit chaotic loss landscapes with large values of the loss and gradient directions that do not point toward the global minima; cf. [104]. The depth of a network is a

key factor in its success; see [173] and references therein. Therefore, it is important to be able to train deep networks. Here, gradient-based methods usually fail to find the global or appropriate local minima. However, there are ways to smooth the loss function and increase its convexity, e.g., by modifying the network architecture, for example, by introducing skip connections; cf. [64, 104].

Due to the non-convexity of the loss function, the results of training depend strongly on the initialization of the model weights. Especially in the case of a highly chaotic loss landscape, local optimization methods cannot effectively minimize such a loss function. Here, the only reasonable option is to train multiple models with different random initializations of the weights and hope that one model is able to reach a global minimum [52].

3.4.2 Choice of Activation Function

There are a number of problems associated with the choice of the activation function, and many of the activation functions have been designed to combat specific problems. These include the vanishing and exploding gradient problems [49, 60], non-differentiability at certain points, and the dying ReLU problem [117].

Vanishing gradients mean that the gradients of the network weights become vanishingly small during training, effectively stopping the neural network from training. This problem typically occurs with activation functions that have gradients in the range $(0, 1]$. Because gradients are computed by extensive use of the chain rule, the gradients of early layers in a deep network are computed by multiplying many small numbers. This means that the gradients of early layers decrease exponentially with the depth of the network, slowing down its training. The exploding gradient problem is very similar, but with exponentially growing gradients.

Dying ReLU can be seen as a special case of vanishing gradients. Here, some weights of a layer have been pushed into a state where the input to the activation function of one

or more nodes is less than 0 for almost all inputs. In this state, the affected units become inactive and output only 0 for most inputs. In addition, the gradient of ReLU is 0 in this state, and thus the affected nodes are stuck, reducing the capacity of the model.

4 Neural Networks as Computational Fluid Dynamics Surrogate Models

We have discussed machine learning models and neural networks explicitly as tools for solving a regression problem – that is, as a general function approximation. In particular we are interested in surrogate models that approximate the solution functions for a class of boundary value problems. To this end, we first provide a brief introduction to surrogate models, in particular neural networks, in the context of Computational Fluid Dynamics (CFD) and place them in a broader context. We then discuss data-based models in section 4.2. Here, we specifically discuss models based on Dense Neural Network (DNNs) in section 4.2.1 and Convolutional Neural Networks (CNNs) in section 4.2.2. Following this, we discuss physics-based models in section 4.3 and provide a brief overview of physics-informed DNNs in section 4.3.1. Before turning to physics-aware CNNs as the main focus of this work in chapter 5, we discuss the generation of training data for CNNs as surrogate models for CFD in general in section 4.4, and the CNN architecture we use in this work in section 4.5.

4.1 Surrogate Models

In general, a surrogate model is an approximation of a more complex model or system. Surrogate models are used when the direct use of the complex model is not possible or not feasible. Their evaluation should typically be significantly faster and/or easier than the

evaluation of the complex model while still being sufficiently accurate. Fluid simulations are often extremely time-consuming and computationally expensive, especially in three dimensions. In addition, they must be recalculated for even the smallest changes in geometry or material parameters. This makes CFD applications a prime candidate for surrogate modeling.

In the context of this work, we are particularly interested in surrogate models that are able to handle variations in the geometry. Broadly speaking, we are looking for a surrogate model that takes a representation of a geometry as input and generates a representation of the solution variables of the governing equations as output. This is not the only application of surrogate models, nor is it the only one we are considering. We are also interested in varying boundary conditions and material parameters, although variations in the geometry are our primary point of interest.

In particular, our goal is to train a surrogate model that approximates a solution operator

$$\begin{aligned} U : I &\to O, \\ i &\mapsto o, \end{aligned} \tag{4.1}$$

mapping from an input space I to an output space O, where i is some representation of the input data, such as a pixel image of the geometry in the case of a CNN, and o is some representation of the output data, such as pixel images of the solution variables of the Navier–Stokes equations. Of course, the exact form of the input and output data depends on the operator U we want to approximate and on the surrogate model we choose. This process of training a surrogate model to approximate a solution operator can also be referred to as operator learning. In the following, however, we will use the term surrogate model rather than operator learning.

There is a wide variety of surrogate models, and neural networks are just one possible variant. Among others, there are reduced order models (ROM), for example based on proper orthogonal decomposition (POD) [82], principal component analysis (PCA) [94],

or reduced basis (RB) [14]; generalized Bayesian approaches [146]; or support vector machines (SVM) [29]. In this work, however, we generally focus on neural networks and do not discuss other models. Furthermore, there is a wide variety of neural networks. The boundaries between different architectures are not necessarily well defined, as there are hybrid architectures and smooth transitions. Besides the architectures we covered in chapter 3, there are, for example, recurrent neural networks (RNN) [71], graph neural networks (GNN) [162], spiking neural networks (SNN) [119], and long short-term memory (LSTM) models [71].

Most of the neural network methods mentioned above do not or cannot generalize to previously unseen geometries, or at least have great difficulty in doing so. Of course, these architectures also have their advantages, e.g. LSTMs are well suited to model time-dependent processes; cf. [197]. Therefore, in this work we focus on CNNs, in particular fully convolutional neural networks; cf. section 3.3. As we have already mentioned, CNNs are particularly well suited to handle variations in geometry, in fact they were designed with this explicit purpose in mind [98]. However, for the sake of completeness, we also discuss dense neural networks and related architectures as surrogate models. For an overview of deep learning in CFD we refer to [20].

The use of neural networks as surrogate models for CFD simulations can be considered part of the field of Scientific Machine Learning (SciML) [6, 181], a new and rapidly developing field of research in artificial intelligence. Therefore, all the methods discussed in this chapter can be understood as part of SciML. This field combines scientific computing and machine learning techniques to produce more accurate and interpretable algorithms. This is achieved by integrating domain knowledge into ML algorithms. In this context, domain knowledge may include, for example, physical laws or principles, expert knowledge, or the results of numerical simulations.

Here we consider a division of surrogate models into two categories depending on how they are trained. The first category involves training the models on a data set that

includes reference simulations, and we refer to these as data-based surrogate models. The second category involves training the models using the governing equations or underlying physics. There are many terms used in the literature, such as physics-informed, physics-based, physics-aware, or physics-constrained. Some terms, such as physics-informed, are associated with a particular network architecture; cf. section 4.3.1. Therefore, in this work we will refer to DNNs trained using governing equations as physics-informed neural networks and CNNs trained using governing equations as physics-aware convolutional neural networks.

Both types of models can be seen as supervised regression problems in the context of neural networks. However, the categorization is more obvious for data-based models than for physics-aware models, which we will discuss briefly. For physics-aware models, we do not have pairs of input and output data to which we fit our model. Instead, we only have input data, which we use to generate a physical residual using a procedure that is yet to be defined. We then minimize this residual. However, this abstract process does not correspond to the definition of an unsupervised model. Moreover, the physical residual can be viewed as a kind of prediction of the model that we want to fit to a target value of zero. Therefore, we can treat and implement physics-based models in the same way as supervised models.

4.2 Data-Based Models

In this section we will discuss how we can train a neural network in a data-driven manner to act as a surrogate model for CFD simulations. In general this involves defining the input and output spaces I and O in eq. (4.1) and creating a training data set consisting of pairs of input data i and corresponding output data o.

In this case we take as output data the result of CFD simulations, i.e., a suitable representation of the solution variables for the selected NN architecture. Correspondingly, we choose as input data the input parameters of the CFD simulations, which are varied,

e.g., a representation of the geometry, the boundary conditions, or the material parameters. Of course, it is also possible to use experimental measurements as reference output data instead of numerical simulation results. In addition, the exact form of the input and output data depends on the chosen NN architecture. For example, DNNs are classically pointwise, i.e., the coordinates themselves must be part of the input, and the corresponding output is the solution variables at those coordinates. CNNs, on the other hand, are based on data with a tensor product grid-like topology, i.e., pixel images in 2D, so both the input and the output should consist of pixel images or similarly structured data.

Regardless of the chosen NN architecture, data-based NNs are trained by minimizing a loss function that depends on the predictions of the model and the reference data; c.f. section 3.1. For example, if we choose the mean squared error (MSE) as the loss function l, then training an NN f^Ψ on a data set T, consisting of data pairs (i, o), is given by the minimization problem

$$\arg\min_{\Psi} \frac{1}{|T|} \sum_{(i,o)\in T} \left(f^\Psi(i) - o\right)^2. \tag{4.2}$$

4.2.1 Dense Neural Networks

A DNN takes as input a finite-dimensional vector $x \in \mathbb{R}^n$. Since the connections from the input to the first hidden layer are dense, it makes sense to keep the dimension of the input vector small. For example, a DNN with m nodes in the first hidden layer has $n \cdot (m + 1)$ parameters in this layer. If n is very large, either the first hidden layer must consist of very few units, which would drastically limit the approximation capacity of the network, or the number of parameters would explode, making it expensive and time-consuming to train the DNN. In particular, it is not practical to provide a DNN with a pixel image as input. Even for a pixel image consisting of $128 \times 128 = 16\,384$ pixels, a first layer consisting of only $m = 100$ units will have more than 160 000 parameters. Therefore, DNNs are usually not applied to pixel images, but to a single geometry. The coordinates are the input, and the solution variables at these coordinates are the corresponding output.

Thus, in the context of the two-dimensional Navier–Stokes equations, the input to a DNN consists of the spatial coordinates x and y – and in the case of a time dependence, the time t; and the output consists of the velocity in the x and y directions, u and v, respectively, and the pressure p at the given coordinates. A training data set T then consists of N data pairs of coordinates $\left(x^i, y^i\right)$ and reference values for the solution variables at these coordinates $\left(u^i, v^i, p^i\right)$. We denote the output of this DNN by u^Ψ, v^Ψ, and p^Ψ. We can then train the DNN by minimizing the mean squared error

$$\frac{1}{N}\sum_{i=1}^{N}\left(\left(u^\Psi(x^i,y^i)-u^i\right)^2+\left(v^\Psi(x^i,y^i)-v^i\right)^2+\left(p^\Psi(x^i,y^i)-p^i\right)^2\right). \tag{4.3}$$

Of course, such an NN, where the input consists only of the spatial coordinates, cannot handle variations in the geometry or the boundary conditions. Therefore, it can only be trained on a single geometry for a given boundary value problem. Thus, such a model is not useful as a surrogate model in the context in which we are interested. However, it may be useful if we can use this model via some other techniques to estimate parameters, for example, by incorporating physical constraints. A physics-based loss could be used for this purpose. We will discuss DNNs and physics-based loss functions in section 4.3.1.

For the reasons given above, it is rare for a DNN to be used as a surrogate model in the way we have defined it in section 4.1. Nevertheless, data-based DNNs have been used in the past in many areas of modeling fluid flow problems. For example, to reconstruct turbulent flows [127], to predict surface pressure over time in aeronautics [165], or to improve simulations of turbulent flows [109].

However, there are modifications and special network architectures that allow us to incorporate variations in, e.g., material parameters even with a DNN architecture. These are, for example, universal solution manifold networks (USM-Nets) [147], deep operator networks (DeepONet) [116], and, more recently, neural operators [89]. For example, in [22], neural operators were used as digital twins for an offshore structure under irregular waves. Nevertheless, most of these architectures do not solve the previously discussed

problems of DNNs with respect to the processing of varying geometries. And those that do are severely limited in their capacity. However, since these architectures are important developments in the field of operator learning and surrogate modeling with neural networks, we briefly describe them.

One way to extend a DNN to act as a surrogate model for multiple geometries is to use a low-dimensional parameterization of the geometry as an additional input. USM-nets [147] are an example of this. Here, the geometry is parameterized by so-called landmarks, e.g., coordinates of points of interest. These landmarks are then used as an additional input to a DNN. However, only limited variation in geometry can be handled by such an approach. To handle a large variation in geometry, the dimension of the parameterization would have to become very large. This would cause the problems of DNNs with too large inputs described above to reappear, and the training and evaluation of the model would become inefficient. On the other hand, if the dimension of the parameterization is kept small, it may not be able to adequately represent the variation in the geometry, causing the model to misinterpret the geometry and make incorrect predictions.

DeepONets [116] and Neural Operators [89] are special architectural extensions for, among others, DNNs that enable NNs to be used as surrogate models for solution operators. A DeepONet consists of two subnetworks, a branch network and a trunk network, and maps from a representation of an input function f and a coordinate x to the solution operator U evaluated for the input function f at the coordinate x, i.e., $U(f)(x)$. Here, the branch network takes as input a finite representation of the input function f, for example a material parameter function, evaluated at a given set of points, and the trunk network takes as input the coordinate x at which we want to evaluate the solution. The outputs of the branch and the root networks are then combined in a linear combination. One advantage of a DeepONet is that the points at which the input function is to be evaluated do not have to adhere to a fixed structure. Another advantage is that the DeepONet can theoretically be trained and evaluated at any coordinate. DeepONets are

not limited to DNNs as their subnetworks. However, using convolutional layers in the branch network would again impose structure on the points where the input function is evaluated. With neural operators, typical fully connected layers are extended by an additional application of a specific operator, e.g., a transformation into Fourier space, cf. [108], application of an integral kernel operator, and finally a projection back into standard space. For a comprehensive overview of DeepONets and neural operators, we refer to [39, Sect. 5].

4.2.2 Convolutional Neural Networks

CNNs are particularly good at processing image data. Therefore, a CNN is especially well suited to work as a surrogate model between image spaces. Thus, the solution operator that we want to approximate assumes an input space I, which consists of images of the geometry, and an output space O, which consists of images of the solution variables. In particular, let $I_g \in \mathbb{R}^{W \times H}$ be the matrix resulting from a pixel image representation of some computational domain Ω_g, where W and H denote the width and height of the pixel images, respectively. Furthermore, let u_g, v_g, $p_g \in \mathbb{R}^{W \times H}$ be matrix representations of the solution variables of the Navier–Stokes equations. Then the solution operator we want to approximate is given by

$$\begin{aligned} U : \mathbb{R}^{W \times H} &\to \mathbb{R}^{3 \times W \times H}, \\ I_g &\mapsto (u_g, v_g, p_g)\,. \end{aligned} \tag{4.4}$$

We now replace U with a CNN denoted by U_{NN}^{Ψ}, where Ψ denotes the trainable network parameters. Specifically, U_{NN}^{Ψ} is defined as

$$\begin{aligned} U_{NN}^{\Psi} : \mathbb{R}^{W \times H} &\to \mathbb{R}^{3 \times W \times H}, \\ I_g &\mapsto \left(u^{\Psi}(I_g), v^{\Psi}(I_g), p^{\Psi}(I_g)\right). \end{aligned} \tag{4.5}$$

Thus, for a training data set T consisting of data pairs $(I_g, (u_g, v_g, p_g))$ for different geometries g, the data-based loss is given by

$$\frac{1}{|T|} \sum_{g \in T} \|u^\Psi(I_g) - u_g\|_2^2 + \|v^\Psi(I_g) - v_g\|_2^2 + \|p^\Psi(I_g) - p_g\|_2^2. \tag{4.6}$$

Note that we have provided a three-term loss function here, but it is also possible to train a CNN as a surrogate model for, e.g., only the velocity components u_g and v_g, instead of all three solution variables. Furthermore, there are many variations of the input space I in the literature, with the use of a pixel image of the geometry as input being the most useful in our case, as it allows the model to handle variations in the geometry. Note also that in this case both the input and the output are discretizations of the original geometry and solution variables. Therefore, it is expected that some information about the geometry and the solution variables will be lost.

CNNs are very popular as surrogate models for fluid flow. Consequently, there is an abundance of available publications. We do not give a detailed literature review here, but only mention some examples. For example, CNNs have been used as surrogate models for the velocity components of fluid flow in a channel around an obstacle [35, 36, 58]. This approach has also been extended to include pressure as an additional output, for example in [152]. The architecture used there is very similar to the one we use; cf. section 4.5. Also, the model problem studied there is the inspiration for the flow in a channel model problem we consider; cf. section 2.3.

CNNs have also been applied to many different geometries such as porous media in two dimensions [178] as well as three dimensions [160]. Furthermore, there are approaches that do not take the geometry as input, but for example a time series of the pressure, as in [77] for the case of flow around a cylinder. In addition, CNNs have been used to advance and improve fluid simulations [67, 183, 190]. There are numerous other applications of CNNs related to fluid flow, such as a particle based surrogate model using continuous convolutions [184], smoke simulations [26, 84], aerodynamics [9, 166],

Bayesian convolutional autoencoders for uncertainty quantification [210], general model order reduction [42, 99, 121], and many more.

4.3 Physics-Informed / Physics-Aware / Physics-Based Models

The general concept of using prior knowledge of physics phenomena with ML models is known as physics-informed ML. In this work we focus on methods that involve constructing a physics-informed loss function based on the governing equations. For a general overview of physics-informed ML we refer to [80, 125, 201].

Training an ML model by minimizing a physics-informed loss function results in a model that approximates the solution of the governing equations while also, in contrast to data-based models, enforcing physical plausibility in its predictions. Explicitly, no reference data, i.e. simulation results or measurements, are required. Thus, the model is trained using only our knowledge of physics and without training data. Nevertheless, it is possible to include available reference data in the training process by combining the data-based and the physics-informed loss. It remains to be determined how to construct the physics-informed loss. In the following, we examine two methods.

One of them is the widely used physics-informed neural network (PINN) approach, first explored in the 1990s [33, 51, 93] and recently rediscovered in [142, 143]. Here, the computation of the physics-informed loss is based on automatic differentiation (AD) and as such requires the spatial (and temporal) coordinates to be part of the input. It is commonly used with DNNs and similar architectures. Therefore, we will discuss physics-informed neural networks using a dense neural network as an example, see section 4.3.1.

Another method is the physics-aware CNN approach. Here, a physics-aware loss is constructed based on image data and using finite difference filters [168]. We will discuss this method in more depth in chapter 5. To the best of the author's knowledge there

is no comprehensive review literature on physics-aware CNNs. However, physics-aware CNNs are mentioned in [30, Sect. 2.1.2].

4.3.1 Physics-Informed Dense Neural Networks

A physics-informed neural network (PINN) [142] is a neural network that is trained by minimizing a physics-informed loss function. This definition is quite flexible and could include physics-aware CNNs. In the literature, however, the term PINN is usually used only to refer to NNs for which the derivatives required for the physics-informed loss are computed using automatic differentiation (AD). Therefore, we consider the term PINN to be restricted to network architectures that have the spatial (and temporal) coordinates as input and the solution variables at these coordinates as output, since this architecture is necessary to compute the spatial (and temporal) derivatives using AD. In particular, this is often the case for DNNs, cf. section 4.2.1.

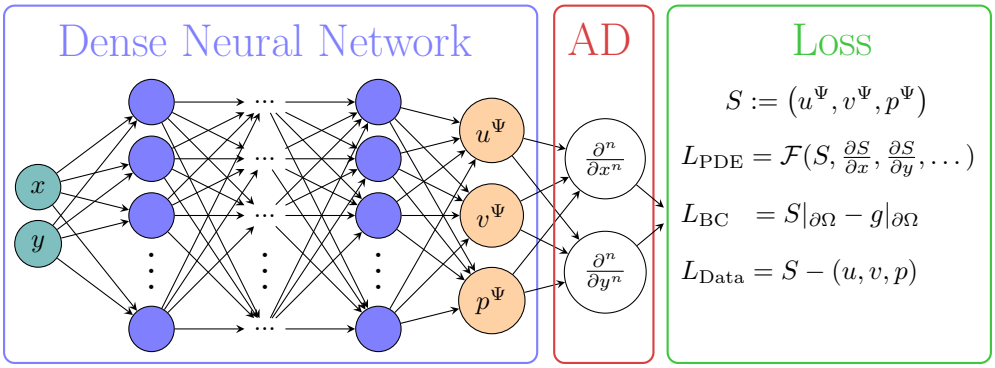


Figure 4.1: Schematics of a physics-informed neural network, inspired by [19]. $\mathcal{F}$ represents a PDE, for example the Navier–Stokes equations.

The concept of using AD to compute derivatives and using them to train an NN to satisfy a PDE was discovered as early as the 1990s [33, 51, 93]. However, it has recently been rediscovered [142, 143] and has subsequently become widely popular.

We give a short example of a PINN. Again, consider a DNN as in section 4.2.1. Then, for the steady state incompressible Navier–Stokes equations, and the reference solution data u, v and p, a simplified scheme of a PINN with the individual loss terms is shown in fig. 4.1. In particular, the physics-informed loss of a PINN is a weighted sum of the individual loss terms shown in fig. 4.1

$$L = \omega_1 \cdot L_{\text{PDE}} + \omega_2 \cdot L_{\text{BC}} + \omega_3 \cdot L_{\text{Data}}, \tag{4.7}$$

where L_{Data} is the data-based loss eq. (4.3) from section 4.2.1, L_{BC} is a data-based loss specifically on the boundary $\partial\Omega$, and L_{PDE} is

$$\frac{1}{N_f} \sum_{i=1}^{N_f} \left\| \left((\vec{u}^{\Psi} \cdot \nabla)\vec{u}^{\Psi} - \nu \Delta \vec{u}^{\Psi} + \nabla p^{\Psi} \right) \right\|_2^2 + \left\| \nabla \cdot \vec{u}^{\Psi} \right\|_2^2, \tag{4.8}$$

where all partial derivatives are calculated using AD.

PINNs have two main advantages. First, they do not rely on reference data. Except for the boundary conditions, which still need to be specified, a PINN can be trained completely without reference data. Second, PINNs can be applied to inverse problems even with a limited amount of available data.

However, PINNs also have disadvantages. Since PINNs are based on the same network architectures as those discussed in section 4.2.1, namely DNNs, they share their advantages as well as their limitations. For example, a PINN can be trained at any point and is thus mesh-free. However, using a DNN or related architectures drastically limits the variation in input that the model can handle. In particular, a classic PINN is essentially incapable of handling variations in geometry. As before, modifications to the NN architecture are necessary to extend the capabilities of a PINN as a surrogate model. Examples include geometry aware physics informed neural networks (GAPINNs) [134], where a low dimensional representation of the geometry is learned via a convolutional autoencoder, and physics-informed PointNets [79], an NN architecture consisting of multiple subnetworks that processes unordered point clouds. Moreover, the inclusion of the boundary conditions

via an additional term in the loss function, cf. eq. (4.7), is one of the weak points of PINNs, since it introduces an undesirable trade-off between the satisfaction of the governing equations and the boundary conditions, cf. [176]. Note that this problem can be circumvented by enforcing the boundary conditions using a distance function [175]. However, this solution increases the dependency on the given geometry. There are also physics-informed DeepONets [191] and physics-informed Neural Operators [106] that extend the capabilities of basic PINNs. In addition, basic PINNs are slower than classical numerical solvers for forward problems, cf. [115]. Also, PINNs have difficulties learning correct predictions for convection dominated problems, cf. [21].

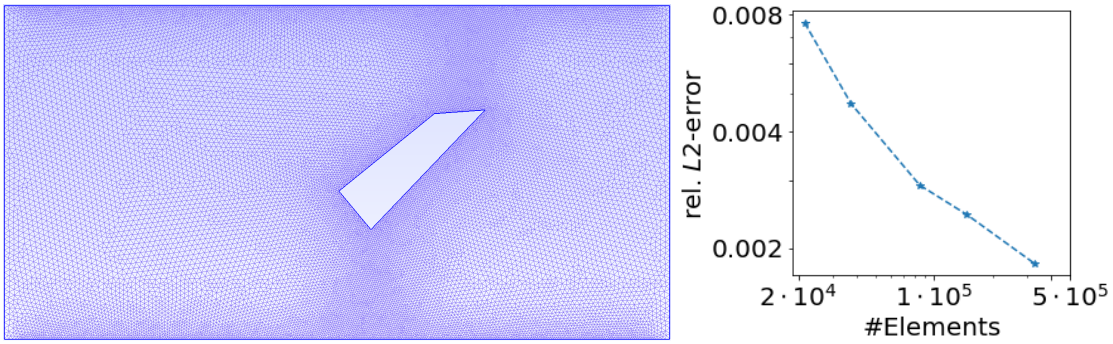
For a comprehensive overview of PINNs we refer to [30, 39] and for their use in fluid mechanics we refer to [19]. There is also a comparison of network architectures with respect to PINNs applied to three-dimensional blood flow available in [130].

4.4 Generation of Training and Validation Data

In this section, we describe the process of creating the data sets that we will later use to train and validate convolutional neural networks; cf. chapter 6 and chapter 7. We describe the meshes in section 4.4.1, and then the simulations we use to obtain reference solutions in section 4.4.2. Following this, in section 4.4.3 we describe the process by which we create the pixel images of the geometries as well as the reference solutions. For this purpose, we consider two-dimensional and three-dimensional channel geometries as examples, see section 2.3.

4.4.1 Meshes

For the construction of the meshes we use Gmsh [46]. For a detailed description, please refer to its documentation. For mesh generation, we use the Frontal Delaunay algorithm for unstructured triangular meshes. We refine the mesh near the no-slip boundary, i.e.,



(a) (b)

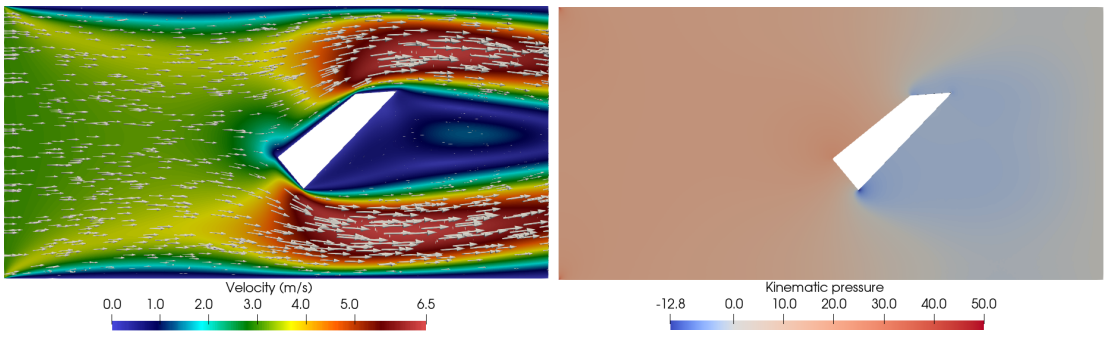
Figure 4.2: (a) Example of a mesh used for simulations and (b) a mesh convergence plot.

near the top and bottom walls and near the obstacle, to allow for proper boundary layers to form. See fig. 4.2(a) for an example of a two-dimensional mesh. This mesh is rather coarse, consisting of only about 40 000 elements, while we used meshes with about 160 000 - 200 000 elements for the simulations. To ensure that the mesh resolution is sufficient, we performed a mesh convergence study. For a channel geometry with a medium sized obstacle, we computed a reference solution on a very fine mesh consisting of about 1 300 000 elements. The relative error compared to the reference solution is shown in fig. 4.2(b) for finer meshes. Similar studies have been carried out for the other model problems as well, and we consider a mesh to be fine enough if the relative error compared to the fine reference solution is less than 1%. For instance, for the three-dimensional channel geometry, calculations are performed on meshes with $1\,000\,000 - 2\,000\,000$ elements, and the convergence study was performed with a reference solution calculated on a mesh with approx. 6 000 000 elements.

4.4.2 Simulations

Finite volume simulations were performed to obtain reference solutions for the velocity and pressure field values. For this purpose, the open source software OpenFOAM [180]

is used. Explicitly, the Semi-Implicit Method for Pressure Linked Equations-Consistent (SIMPLEC) [138, 187] is used to solve the steady-state incompressible Navier–Stokes equations. Here we use a Geometric Agglomerated Algebraic Multigrid (GAMG) solver [103] to solve the pressure equations and the preconditioned Stabilized Bi-Conjugate Gradient Method (BiCGSTAB) [185] with a Diagonal Incomplete LU preconditioner (DILU) to solve the velocity equations. In particular, we employ bounded second order upwind schemes for the convective terms and standard second order schemes for all other terms. In the inner iterations, we solve for the pressure up to a relative tolerance of 10^{-5} or an absolute tolerance of 10^{-7} and for the velocity up to an absolute tolerance of 10^{-8}. In the outer iterations, we solve up to a relative tolerance of 10^{-4} for the pressure and 10^{-5} for the velocity. A velocity field and a pressure field calculated in this way can be seen in fig. 4.3.



(a) Velocity (b) Pressure

Figure 4.3: Velocity (a) and pressure (b) obtained from an OpenFOAM simulation.

On a single core, the finite volume simulations themselves take between 10 and 60 minutes for two-dimensional cases and between 60 and 300 minutes for three-dimensional cases. The simulations may take longer depending on the complexity of the geometry. Of course, this can be accelerated by utilizing multiple cores.

4.4.3 Image Creation

The models we consider are convolutional neural networks and they work on structured data. Therefore, we generate pixel images of the geometries and the solution variables of the reference solutions. To do this, we place a uniform grid over the geometry and associate the grid nodes with the centers of the pixels. Subsequently, we evaluate the geometry at the grid nodes by checking if the grid node is in the geometry and evaluate the solution variables by interpolating them to the grid nodes. This process is illustrated for the pixel image of the geometry from fig. 4.3 in fig. 4.4.

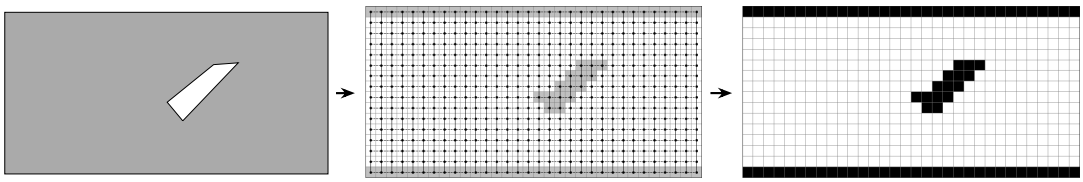


Figure 4.4: Construction of a pixel image of the geometry, here with a resolution of 32×16 pixels.

In general, this process is a bit more complicated, so let us present some more details. Following our approach in the rest of this work, cf. chapter 5, we consider a rectangle Q containing the geometry Ω and decompose it into a uniform grid Q_h. It is of interest that Q is as small as possible and completely contains Ω. In the case of the channel geometries, a suitable Q is easy to find because the geometry Ω is already a rectangle from which an obstacle has been cut. We then associate the grid Q_h with a pixel image by identifying the grid nodes as the centers of the pixels.

There are several ways to assign values to pixels. One straightforward choice is a binary image, i.e., each pixel is assigned a value of 1 if it is inside the geometry, and a value of 0 otherwise. It is also possible to allow a continuum between 0 and 1 instead of just two values, where this number could represent, for example, what percentage of the pixel is

inside the geometry. Another option is the signed distance function, i.e. assigning each pixel a real number corresponding to the distance of that pixel to the nearest point on the boundary of Ω. However, most of these options require exact knowledge of the geometry, and it is usually not possible to achieve this exactness in practice. Therefore, in this work we consider unambiguously segmented binary pixel images. In practice, these can be generated from medical scans, such as Magnetic Resonance Imaging (MRI), using ML models, for example.

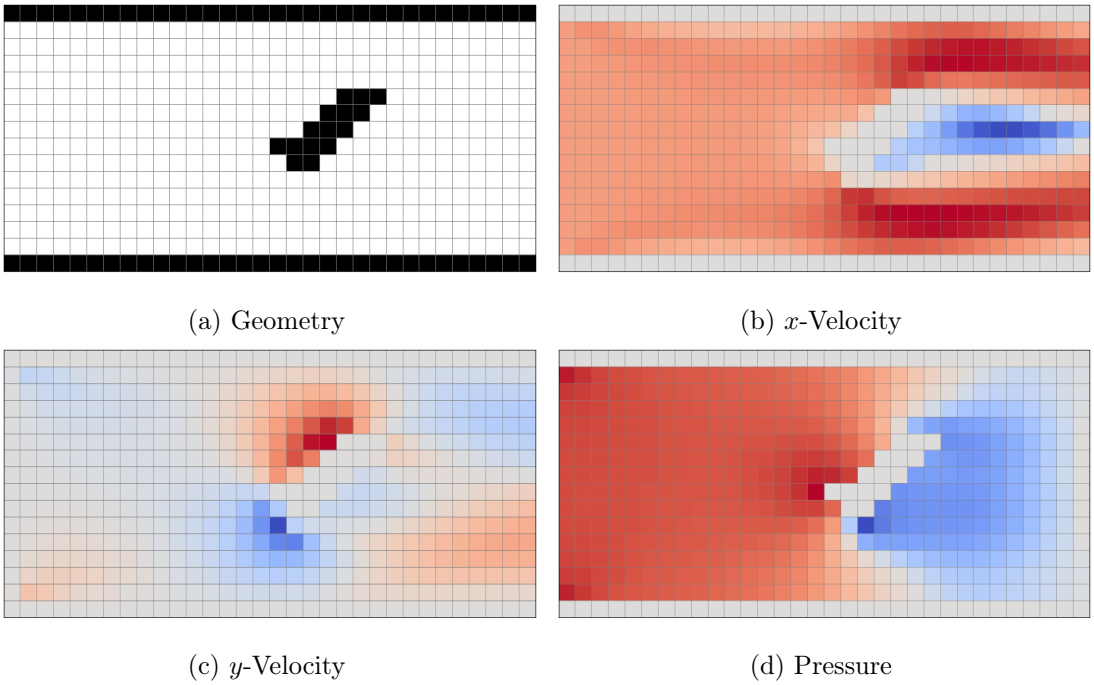
With the decision to use binary pixel images to represent the geometry, the only remaining question is how to determine whether a pixel is assigned the value 0 or 1. The approach we take is to consider a pixel to be inside the geometry if its center is inside the geometry. We also consider a pixel to be inside the geometry if its center is on the inflow or outflow boundary. Correspondingly, we consider a pixel to be outside the geometry if its center is outside the geometry or on the no-slip boundary. However, there are other possible approaches. An alternative approach is to consider whether more than 50% of each pixel is inside the geometry. This approach may lead to different results than the previously discussed approach, especially for particularly jagged boundaries.

Explicitly, when constructing the pixel images of the geometries, we consider the inflow and outflow boundaries as inner parts of the geometry. These areas are basically just treated as a boundary because we can only consider a finite computational domain Ω. In reality, the geometry continues here.

Finally, we require labeled reference data for the solution variables. In general, we need these for evaluating a CNN and for training a data-based or hybrid CNN. A hybrid CNN is trained on both the data-based and the physics-aware loss. In particular, for a physics-aware CNN, we need reference data for evaluation only.

There are several ways to do this. The simplest one, which we use here, is to interpolate the solution variables to the pixel grid. That is, we evaluate the simulation results at the centers of the pixels. Examples of the resulting images are shown in fig. 4.5. Note that

the values of the solution variables in pixels outside the geometry are set to 0. This can be easily achieved by masking the output images based on the geometry representation in the input image. Other approaches include averaging the simulation results over the pixels, also known as Clément-type interpolation.



(a) Geometry (b) x-Velocity

(c) y-Velocity (d) Pressure

Figure 4.5: Exemplary representations of the pixel images, here with a lower resolution of 32×16 pixels.

4.5 Convolutional Neural Network Architecture

We aim to design a surrogate model that maps from a pixel image of the geometry to pixel images of the solution variables. We are inspired by the architectures used in the [35, 36, 58]. The architectures there are based on the U-Net [154].

A U-Net architecture generally consists of an encoder and a decoder. Since we have

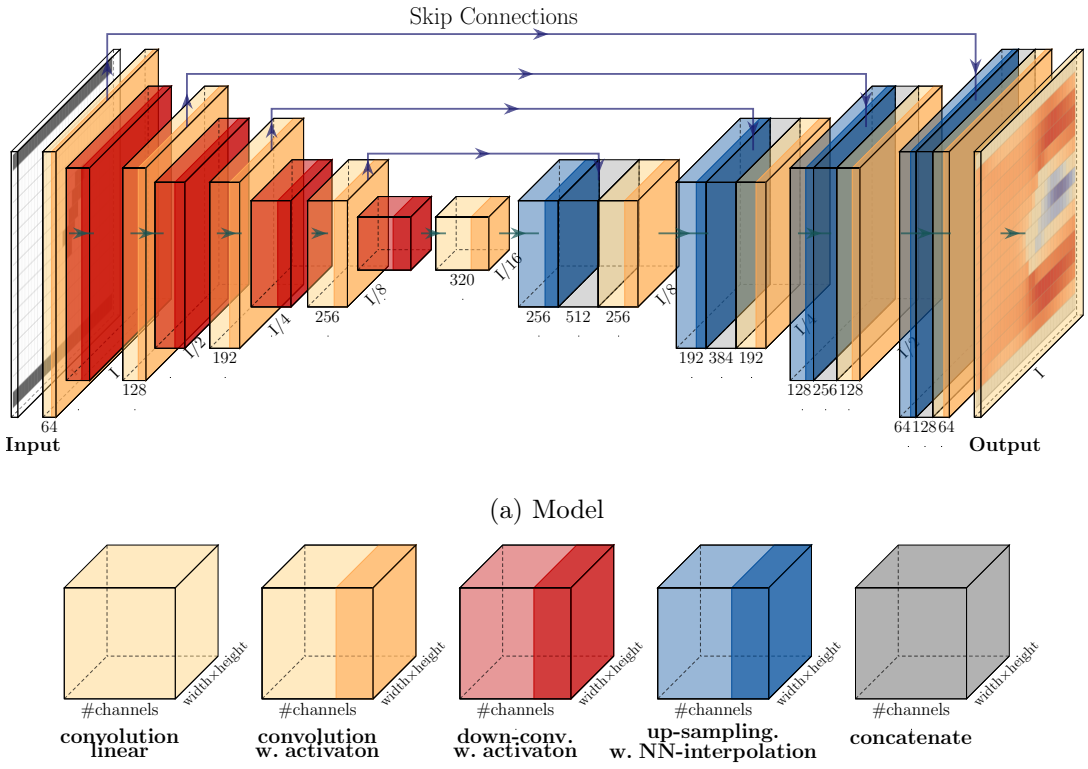
multiple solution variables, the question arises whether we use a common decoder for all solution variables or separate decoders. There is conflicting information in the literature as to whether a common decoder for all solution variables or separate decoders are preferable. There is literature where it was reported that a shared decoder is more performant [9] and literature where it was reported that separate decoders perform better [152]. In this work, separate decoders are used. Thus, our CNN model consists of one encoder and three decoders, one for each solution variable. In three dimensions, our model consists of four decoders.

In the encoder, the spatial dimension is reduced and the feature dimension is increased. In the decoder, this process is reversed, i.e., the spatial dimension is increased and the feature dimension is reduced. The two parts of the model each consist of individual blocks, which we will refer to as encoder-block and decoder-block.

An encoder-block consists of two convolutional layers. The first layer has filters of size 3×3 with stride 1 and the second layer has filters of size 2×2 with stride 2. The first layer fulfills the standard purpose of a convolution, that of extracting features. The second layer, on the other hand, serves the purpose of a pooling layer by reducing the spatial dimension. The use of a convolutional layer instead of a pooling layer, however, reduces the occurrence of high-frequency artifacts, which have been associated with pooling layers; cf. [73]. Apart from that, a convolutional layer with the chosen properties essentially serves the same purpose as a pooling layer, except that the weights it uses can be learned instead of being fixed, as is usually the case with pooling.

A decoder block effectively mirrors an encoder block. Thus, a decoder block consists of an upsampling layer with nearest-neighbor interpolation followed by a standard convolutional layer with filters of size 3×3. With the upsampling layer, we mirror the strided convolution from the encoder block and increase the spatial dimension by a factor of 2. The subsequent standard convolution enhances the expressiveness of the decoder block. We employ an upsampling layer with nearest neighbor interpolation instead of a deconvolutional layer, as

this avoids the checkerboard oscillations associated with deconvolutional layers; cf. [133] and [45]. In all of the convolutional layers, we use ReLU [50] as the activation function by default, and we initialize the weights with the He initialization [63].



(a) Model

(b) Legend

Figure 4.6: Exemplary model architecture of a four-level deep encoder-decoder with skip connections.

A complete model consists of one encoder part, which is composed of nl consecutive encoder blocks, and three decoder parts, each of which is also composed of nl consecutive decoder blocks. Here, the number of levels within our model is denoted by nl. In addition, we connect the encoder and decoder blocks that are at the same level, i.e., whose spatial dimensions match, with a skip connection. To do this, we concatenate the output of the

first convolutional layer of the encoder block to the output of the up-sampling layer of the decoder block. These skip connections allow us to train deeper models [64] and they simplify the loss landscape; cf. [104]. In general, we use a fixed number of filters in the first block of the encoder part and we refer to this number as sc. In each subsequent lower layer, the number of filters increases by sc, such that, for example, in the third layer, each layer has $3 \cdot sc$ filters or channels. We mirror this behavior in the decoder part, so that an encoder and decoder block at the same level use the same number of filters. An example of such a model with $nl = 4$ levels and $sc = 64$ filters in the first layer is shown in fig. 4.6.

To account for the different magnitudes and distributions of the outputs, it is possible to provide the CNN with a mean and standard deviation for each output variable. These values are used to transform the output of the CNN. This allows the CNN to learn each output variable in the same order of magnitude. This concept is commonly known as feature scaling, and the specific scaling technique we use is known as standardization; see [57].

With the preparatory work done in this chapter, we now have everything necessary to present the main method of this thesis in the next chapter. There we describe how a convolutional neural network can be trained as a surrogate model for CFD simulations of incompressible fluids by utilizing only the underlying physics.

5 Physics-Aware Convolutional Neural Networks

This chapter is devoted to the primary method of this work, namely the development of a physics-aware convolutional neural network as a surrogate model for flow in varying geometries. That is, we construct a surrogate model that can be quickly evaluated for a given geometry, with sufficiently accurate predictions. Such a model can also be considered as an operator approximation in this context. As described in the previous chapter, the solution operator this surrogate model approximates is generally of the form

$$\begin{aligned} U : I &\to O, \\ i &\mapsto o. \end{aligned} \tag{5.1}$$

Here, in the case of CNNs, we work with tensor product like objects, explicitly considering pixel images in two dimensions and voxel images in three dimensions. That is, I and O are finite-dimensional spaces whose elements are pixel or voxel images. In this chapter, we consider the two-dimensional case, i.e., pixel images. An extension to three dimensions is straightforward, and we present corresponding results in chapter 7. At this point the exact form of the input space I is of little importance for the method presented in this chapter. Rather, the input only has to be able to represent the considered variation of the BVP in a suitable way. Since we are interested in flows in different geometries, it is only natural that the input should reflect this variation in geometry. And since we are using CNNs as our surrogate model, it is natural to use an image of the geometry as input.

In the case of two-dimensional images that are W pixels wide and H pixels high, the discrete solution operator, which maps a pixel image of the geometry to pixel images of the solution of the corresponding boundary value problem, is given by

$$\begin{aligned} U : \mathbb{R}^{W\times H} &\to \mathbb{R}^{n\times W\times H}, \\ I_g &\mapsto u_g. \end{aligned} \tag{5.2}$$

We now want to approximate this solution operator with a CNN. However, instead of using a data-based loss as in section 4.2.2, we want to use a physics-aware loss that depends exclusively on the governing equations and requires no reference data. Since our CNN maps from image data to image data, we cannot use automatic differentiation to obtain derivatives of the function, as in section 4.3.1. The structure of pixel images, however, allows us to efficiently approximate derivatives using finite differences. This connection of CNNs and finite differences was first leveraged in [168] to train a CNN as a surrogate model for solutions of the stationary diffusion equation for varying boundary conditions in a fixed simple square geometry. There was also an earlier use of finite differences to construct a physics-aware loss; see [33]. But here a DNN was used. This approach was apparently not pursued further.

More recently, the combination of finite differences and CNNs has been applied more often to train surrogate models for PDEs. For example, in [43] CNNs were trained for the incompressible Navier–Stokes equation in parameterized geometries. In [118], the authors also trained CNNs, in particular a U-Net architecture, for the incompressible Navier–Stokes equations. Here, the CNN was trained for different Reynolds numbers on a fixed square geometry with a cylindrical obstacle. Further, in [208] CNNs were trained utilizing techniques from the FVM for two-phase Darcy flows in heterogeneous porous media. In [114, 113], the connection between convolutions and the numerical approximation of differential operators was leveraged to learn PDEs from data. There are more applications of physics-aware CNNs based on finite-difference approximations to solve PDEs [10, 38, 149, 163, 190, 207], upscale and denoise solutions [44, 81], or generally

improve the predictive quality of a model [159, 199].

The rest of the chapter is organized as follows. We discuss the connection between finite differences and discrete convolutions in detail in section 5.1. In section 5.2, we derive a minimization problem for a general PDE that we can use to train a CNN. In section 5.3, we derive this minimization problem explicitly and in more detail for the Navier–Stokes equations. In section 5.4, we address the treatment of boundary conditions in detail before finally explaining how we create a model from the previously discussed minimization problem, in section 5.5.

5.1 Finite Differences and Convolutions

In this section, we illustrate that the use of finite differences in conjunction with CNNs is not only possible, but also practical. We outline the similarity of applying finite differences to a two-dimensional grid function and applying convolutions to a pixel image. Explicitly, on a uniform Cartesian grid, we can always express the application of a finite difference approximation as a convolution on a corresponding pixel image.

For this purpose, we consider the following boundary value problem: Find the function u such that

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= g \quad \text{in } \Omega = [0,1]^2, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{5.3}$$

where g is some suitable right-hand side function. For this BVP we have already derived a linear system of equations in section 2.2.1 using finite differences. For this purpose, we introduced a uniform grid $\Omega_h = \{x_{ij} | 1 \leq i, j \leq n+1\}$ with $x_{ij} = ((i-1)h, (j-1)h)$ and used centered differences to obtain the following system of $(n+1)^2$ equations

$$Au^h = g^h. \tag{5.4}$$

Here, u^h and g^h are grid functions of u and g on the grid Ω_h, i.e. they are vectors with entries $u^h_{i+j\cdot(n+1)} = u(x_{ij})$ and $g^h_{i+j\cdot(n+1)} = g(x_{ij})$. To simplify notation, we write

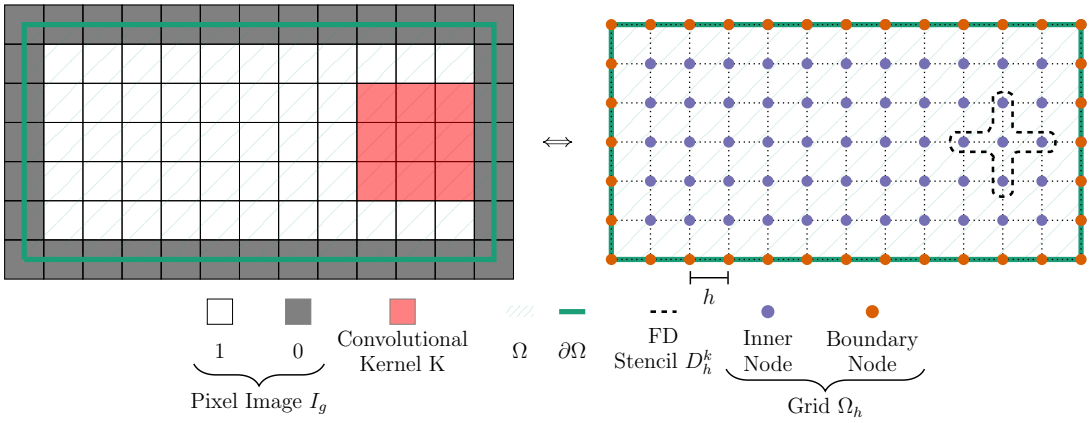


Figure 5.1: A pixel image I_g (left) and the corresponding FD-grid Q_h (right) of a geometry Ω. The boundary $\partial\Omega$ is drawn in green. Note that the values associated with the pixels and their grid node counterparts are not shown. The coloring separates nodes/pixels that lie within the geometry and nodes/pixels that lie on the boundary.

$u^h_{i,j} := u^h_{i+j\cdot(n+1)}$, but u^h remains a vector. Then the individual rows of eq. (5.4) are given by

$$\frac{u^h_{i-1,j} + u^h_{i,j-1} - 4u^h_{i,j} + u^h_{i+1,j} + u^h_{i,j+1}}{h^2} = g^h_{i,j} \quad \forall 1 < i,j < n+1, \tag{5.5}$$

with $u^h_{i,j} = 0$ for $i,\, j \in \{1, n+1\}$. An abstract representation of this mesh can be seen on the right side of fig. 5.1.

Let us now recall the definition of a convolution as it is commonly implemented in contemporary ML frameworks; cf. section 3.3. Let $I \in \mathbb{R}^{W\times H}$ be a two dimensional image and $K \in \mathbb{R}^{k\times l}$ a two dimensional kernel. Then a convolution is given by

$$(I * K)_{i,j} = \sum_m \sum_n I_{i+m,j+n} K_{m,n}. \tag{5.6}$$

Hence, to be able to express the system of equations in 5.4 using convolutions, we need to represent the grid functions u^h and g^h as matrices and define a suitable kernel K. To

this end, we define two matrices $U^h \in \mathbb{R}^{(n+1)\times(n+1)}$ with $U^h_{ij} = u^h_{i,j}$ and $G \in \mathbb{R}^{(n+1)\times(n+1)}$ correspondingly with $G^h_{ij} = g^h_{i,j}$. Finally, we define the kernel $K \in \mathbb{R}^{3\times 3}$ as

$$K = \frac{1}{h^2}\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}. \tag{5.7}$$

As remarked in section 3.3, when a convolution is applied, the size of the output is reduced relative to the size of the input by a factor determined by the size of the kernel. In this case, the kernel is of size 3×3, and the output thus is of size $(n-1)\times(n-1)$. Conveniently, the grid nodes for which there is no counterpart in the output of the convolution are the boundary nodes. There, we enforce the boundary conditions anyway. In this section we restrict ourselves to the interior nodes, which we denote here by $\mathcal{I}$. We discuss how we enforce boundary conditions in the context of CNNs in section 5.4.

We can now observe that

$$\Big(Au^h\Big)|_{\mathcal{I}} = g^h|_{\mathcal{I}} \quad \Longleftrightarrow \quad U^h * K = G^h|_{\mathcal{I}} \tag{5.8}$$

holds. Here $\big(Au^h\big)|_{\mathcal{I}} = g^h|_{\mathcal{I}}$ are the $(n-1)^2$ equations at the inner grid nodes, and $G^h|_{\mathcal{I}}$ is a $(n-1)\times(n-1)$. Thus, we have shown that a finite difference discretization of a PDE on a uniform grid can be implemented using convolutions in a CNN without difficulty. Analogously, we can show that for any finite difference stencil, there is a filter K such that applying the stencil to a grid function, defined on an uniform grid, is equivalent to applying a convolution with the filter K to a corresponding pixel image.

Non-Rectangular Geometries In eq. (5.8) we exploited that Ω is a rectangular domain. Thus the nodes dropped by applying convolution are exactly the nodes on the boundary. But what happens for domains that are not rectangular? Here an extended procedure is required, which is essentially a generalization of the approach for rectangular areas.

In the case of non-rectangular geometry Ω, we define a rectangle Q encompassing Ω and discretize this rectangle with a uniform Cartesian grid Q_h; cf. fig. 5.2. In this case,

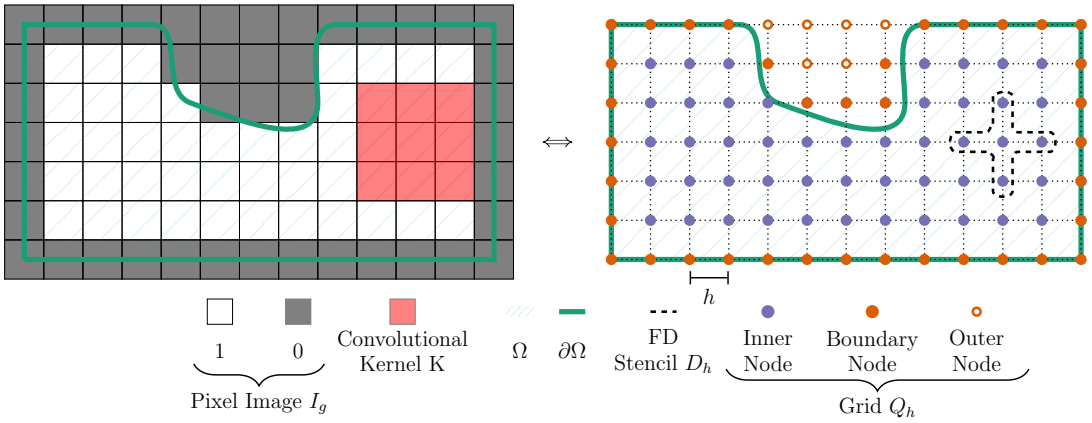


Figure 5.2: A pixel image I_g (left) and the corresponding FD-grid Q_h (right) of a non-rectangular geometry Ω.

we define the grid functions u^h and g^h as well as the matrices U^h and G^h on Q_h instead of Ω_h. Now we have to specify how to handle nodes that are not in Ω. Here we distinguish between nodes that have neighbors in Ω and those that do not. The former are counted as boundary nodes, which we denote by $\mathcal{B}$, while the latter are counted as outer nodes, which we denote by $\mathcal{O}$. We set the values of all grid functions and matrices belonging to the outer nodes $\mathcal{O}$ to 0. In addition, no equations are imposed at the outer nodes, since the governing equations do not apply here. Thus, in the case of non-rectangular geometry, a finite difference discretization of a PDE on a uniform grid can also be implemented using convolutions in a CNN. How exactly we deal with this problem in practice will be discussed in section 5.4.

5.2 Minimization Problem for a Generic PDE

In this section, we derive a minimization problem for a generic PDE using finite differences. This minimization problem can then be used to train a CNN as a surrogate model for the considered PDE without reference data. To do this, we use the relation studied in

detail in section 5.1 between the output of a CNN, which is a pixel image of a solution, and a finite difference solution on a uniform grid. Explicitly, we use finite differences to approximate the required derivatives and thus approximate the residual of the considered PDE. We then formulate a minimization problem over a loss function consisting of this residual. In doing so, our approach is initially decoupled from CNNs – to illustrate that this minimization problem is not necessarily restricted to only training CNNs we first derive it in general.

Let us consider a generic system of PDEs, given in implicit form, on a computational domain $\Omega \subset \mathbb{R}^2$

$$\begin{aligned} F\left(x,\ f(x)\,,\ Df(x)\,,\ D^2 f(x)\,,\ \dots\right) = 0, \qquad & x \in \Omega, \\ B\,(x) = b, \qquad & x \in \partial\Omega. \end{aligned} \tag{5.9}$$

In this case, f is an arbitrary function with sufficient regularity, $D^k f$ are the partial derivatives of f of order k, and b is a function supplying fitting boundary conditions. In order to discretize this PDE using finite differences, we first need to discretize Ω by a mesh. To be able to introduce a CNN as a surrogate model later, we require a uniform mesh. For this purpose, we consider a rectangle Q enclosing Ω and discretize it with an equidistant grid Q_h with grid step size h and W nodes in x direction and H nodes in y direction; cf. fig. 5.2 in section 5.1. We denote the set of grid nodes by $X = \{x_{i,j}\}$. For simplicity, we assume that each node is either in Ω or on $\partial\Omega$. This assumption does not hold for all nodes, and we will discuss how to handle nodes that are outside of Ω later in this section. We will also discuss the practical implementation of this procedure explicitly for the Navier–Stokes equations in section 5.4.

We now discretize eq. (5.9) by replacing the derivatives with finite differences. For this purpose we denote with $f_h(X) \in \mathbb{R}^{W\times H}$ the discrete solution vector and with $D_h^k f_h$ a finite difference approximation to the k-th partial derivative. Thus we obtain a system of, not necessarily linear, equations

$$F\left(X,\ f_h(X)\,,\ D_h f_h(X)\,,\ D_h^2 f_h(X)\,,\ \dots\right) = 0, \qquad x \in \Omega. \tag{5.10}$$

Up to this point, this procedure corresponds to the application of the finite difference method; cf section 2.2.1. Accordingly, the boundary conditions are typically incorporated directly into this system. We do not discuss boundary conditions further in the remainder of this section. The treatment of boundary conditions will be discussed in detail section 5.4.

We now transform the system of equations eq. (5.10) into a least-squares problem of the discrete residual

$$\underset{f^h \in \mathbb{R}^{W \times H}}{\arg\min} \left\| F\left(X,\ f_h(X)\,,\ D_h f_h(X)\,,\ D_h^2 f_h(X)\,,\ \dots\right) \right\|_2^2 . \tag{5.11}$$

Let us note that the solutions of eq. (5.10) and eq. (5.11) are identical. However, solving eq. (5.10) is often beneficial for classical numerical solvers, whereas the minimization problem eq. (5.11) is more suitable for a neural network approach. Indeed, by our choice of Q_h as a uniform Cartesian grid, it is possible to represent f_h as the output of a CNN.

Therefore, we replace $f_h\,(X)$ with a CNN, which we denote by $f_{NN}^{\Psi}\,(I_\Omega)$, where Ψ denotes the trainable network parameters. Here, consistent with our notation in section 4.2.2 and eq. (5.2), as well as our approach in section 5.1, we have replaced X with I_Ω, a pixel image representation of the computational domain Ω, and all finite difference approximations with convolutions with corresponding kernels. We continue to use the notation D_h for the corresponding kernels. Hence, we obtain the following minimization problem:

$$\underset{\Psi}{\arg\min} \left\| F\left(I_\Omega,\ f_{NN}^{\Psi}(I_\Omega)\,,\ f_{NN}^{\Psi}(I_\Omega) * D_h,\ f_{NN}^{\Psi}(I_\Omega) * D_h^2,\ \dots\right) \right\|_2^2 . \tag{5.12}$$

We can now use this minimization problem to train a CNN for a fixed computational domain Ω, or a fixed geometry. If we want to train our CNN for a different domain, we only have to change the geometry and enforce suitable boundary conditions on its boundary.

Moreover, we have previously established that there can be nodes of the Q_h grid, or pixels of the image I_Ω, which are not in the geometry Ω. It is clear that the PDE given in eq. (5.9) is defined only in Ω, and therefore we need to consider eq. (5.12) only in pixels whose associated node is in Ω. This is possible, since the residual at the i-th pixel can be

computed independently of all other residuals. In practice, we first compute the residual at all pixels and in a second step set the residual to zero at all pixels whose associated node is not in Ω. This procedure will be discussed in detail in section 5.4.

In the minimization problem eq. (5.12) we implicitly used a loss function, which we want to define in ML terms at this point. According to its definition, a loss function l assigns a score s to a data pair (x, y); cf. eq. (3.2) in section 3.1. In general, x is the input to the ML model, and y is the reference value for the output. In this method, x is the pixel image of the geometry I_Ω, l is the sum of squared error (SSE), and y is simply the zero vector. Thus, in formal terms, our loss function for a single geometry is given by

$$l\left(I_\Omega,\ 0\right) \coloneqq \left\| F\left(I_\Omega,\ f_{NN}^{\Psi}(I_\Omega)\,,\ f_{NN}^{\Psi}(I_\Omega) * D_h,\ f_{NN}^{\Psi}(I_\Omega) * D_h^2,\ \dots\right) - 0 \right\|_2^2 . \tag{5.13}$$

Note that in practice we do not use the SSE as loss function, but rather the mean squared error (MSE). Therefore, the expression in eq. (5.13) would need to be divided by the number of pixels in the image, which is $W \times H$. We omit this for ease of reading. Also, the zero in this loss function can be omitted. However, we include it to emphasize that training a CNN by minimizing this loss function can be interpreted as a supervised regression task. Specifically, we want to fit the residual vector to 0 in every pixel, and especially in those that lie in Ω. Note that in actuality the pixels that are not in Ω have no effect on the loss, as we set the residuals in pixels that are not in Ω to 0; see section 5.4.

However, in this work we are not interested in training a CNN as a model for a single BVP, but as a surrogate model for a certain range of geometries. Therefore, we optimize the loss function eq. (5.13) not only for one geometry, but for a training data set T of geometries g:

$$\arg\min_{\Psi} \frac{1}{|T|} \sum_{g \in T} \left\| F\left(I_g,\ f_{NN}^{\Psi}(I_g)\,,\ f_{NN}^{\Psi}(I_g) * D_h,\ f_{NN}^{\Psi}(I_g) * D_h^2,\ \dots\right) \right\|_2^2 . \tag{5.14}$$

We have thus derived a physics-aware loss function that we can use to train a CNN as a surrogate model to approximate the solution of a generic PDE eq. (5.9) for a variation

of geometries. In the next section, we discuss the derivation of this minimization problem explicitly for the Navier–Stokes equations.

5.3 Application to the Navier–Stokes Equations

By design, the nature of the derivation of the minimization problem eq. (5.14) in the previous section is quite abstract. Therefore, in this section we derive a minimization problem for a specific PDE, the Navier–Stokes equations. We will later use this minimization problem to train CNNs as surrogate models for flow problems in varying geometries. Previously, we have introduced the Navier–Stokes equations in section 2.1 and in terms of the individual components in eq. (2.15) to (2.17), and we have presented a finite difference discretization of them on a uniform grid in section 2.2.2. The approach in this section is very similar to the approach in section 2.2.2, except that we consider two additional steps, i.e., replacing the grid functions of the solution variables with the output of a CNN and defining a minimization problem over a training data set of geometries.

Let $\Omega \subset \mathbb{R}^2$ be a suitable computational domain, Q a rectangular domain enclosing Ω, and Q_h an equidistant grid, discretizing Q with grid step size h and W nodes in x-direction and H nodes in y- direction. Let us note that the step size h of the grid is not necessarily the same in x- and y- direction. For the sake of readability, however, we assume that it is.

Let us again state the Navier–Stokes equations in terms of their components

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} - \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0, \quad \text{in } \Omega, \tag{5.15}$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} - \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) = 0, \quad \text{in } \Omega, \tag{5.16}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \quad \text{in } \Omega, \tag{5.17}$$

where u and v are the x- and y- components of the flow field, p is the pressure, and ν is the kinematic viscosity. Utilizing, for example, centered differences for the first- and

second-order derivatives D_h^x, D_h^y, D_h^{xx}, and D_h^{yy}, we can discretize eq. (5.15) - (5.17) by

$$u_h \circ D_h^x u_h + v_h \circ D_h^y u_h + D_h^x p_h - \nu \left(D_h^{xx} u_h + D_h^{yy} u_h \right) = 0, \tag{5.18}$$

$$u_h \circ D_h^x v_h + v_h \circ D_h^y v_h + D_h^y p_h - \nu \left(D_h^{xx} v_h + D_h^{yy} v_h \right) = 0, \tag{5.19}$$

$$D_h^x u_h + D_h^y v_h = 0, \tag{5.20}$$

for all inner grid nodes. Here $\circ$ is the Hadamard product, that is, the element-wise product. We omit the treatment of boundary conditions for now, and discuss their implementation in detail in section 5.4. Note that we can equivalently express equations 5.18 - 5.20 in terms of convolutions; see section 5.1 for a detailed discussion. We can also express this discretization in terms of a nonlinear system of equations

$$N\left(\vec{u}_h\right) + G\left(p_h\right) = 0, \tag{5.21}$$

$$D\left(\vec{u}_h\right) = 0, \tag{5.22}$$

with a nonlinear operator N, linear operators D and G, the discrete velocity vector field $\vec{u}_h = (u_h, v_h)^T$, and the discrete pressure p_h; cf. section 2.2.2. At this point, we convert this system of equations into a least squares problem of the discrete residuals, as in section 5.2, and obtain

$$\underset{\vec{u}_h, p_h}{\arg\min} \left(\left\| N\left(\vec{u}_h\right) + G\left(p_h\right) \right\|_2^2 + \left\| D\left(\vec{u}_h\right) \right\|_2^2 \right). \tag{5.23}$$

Before we replace $\vec{u}_h$ and p_h with the output of a CNN, we need to briefly consider how the application of the nonlinear operator N to the output of a CNN works. Therefore we split the operator into a nonlinear and a linear part

$$N\left(\vec{u}_h\right) = \vec{u}_h \circ F\left(\vec{u}_h\right) + A\left(\vec{u}_h\right),$$

where F and A are linear operators. Applying F and A to $\vec{u}_h$ is equivalent to applying finite difference stencils to grid functions, and we have demonstrated in section 5.1 that we can represent this by applying convolutions with fixed weights to the output of a CNN.

The operation $\vec{u}_h \circ F\vec{u}_h$ is the element-wise multiplication of two vectors, and this can be represented without problems in the context of CNNs. Therefore, we can now replace $\vec{u}_h$ and p_h with a CNN-based surrogate model.

We define our surrogate model according to the form provided in eq. (5.2), i.e., a CNN f_{NN}^{Ψ} that maps from a pixel image of a geometry to pixel images of the solution of the corresponding boundary value problem

$$\begin{aligned} f_{NN}^{\Psi} : \mathbb{R}^{W\times H} &\to \mathbb{R}^{3\times W\times H}, \\ I_g &\mapsto \begin{pmatrix} u_{NN}(I_g) \\ v_{NN}(I_g) \\ p_{NN}(I_g) \end{pmatrix}. \end{aligned} \tag{5.24}$$

Here, Ψ denotes the trainable parameters of the CNN, I_g is a pixel image representation of the geometry, and $u_{NN}(I_g)$, $v_{NN}(I_g)$, and $p_{NN}(I_g)$ are vectors containing the approximated solution at the grid nodes corresponding to the pixels in I_g. Note that we can also consider the outputs to be matrices, that is, pixel images. Substituting the output of our surrogate model, $\vec{u}_{NN} = (u_{NN}, v_{NN})^T$ and p_{NN}, for $\vec{u}_h$ and p_h in eq. (5.23), we obtain a minimization problem

$$\arg\min_{\Psi} \left(\omega_{\text{Mom}} \| N\left(\vec{u}_{NN}\left(I_g\right)\right) + G\, p_{NN}(I_g)\|_2^2 + \omega_{\text{Mass}} \| D\vec{u}_{NN}(I_g)\|_2^2\right) \tag{5.25}$$

that we can utilize to train our CNN to learn the discrete solution to a BVP on a fixed geometry. Here ω_{Mom} and ω_{Mass} are weights for the two loss terms. As before, however, we are not interested in a model for a single BVP but rather for a variety of geometries. Therefore, we again consider a minimization problem for a training data set T of geometries g

$$\arg\min_{\Psi} \frac{1}{|T|} \sum_{g\in T} \left(\omega_{\text{Mom}} \| N\left(\vec{u}_{NN}\left(I_g\right)\right) + G\left(p_{NN}(I_g)\right)\|_2^2 + \omega_{\text{Mass}} \| D\left(\vec{u}_{NN}(I_g)\right)\|_2^2\right). \tag{5.26}$$

Finally, this loss can be used to train our surrogate model.

5.4 Boundary Treatment

In this section we discuss the treatment of boundary conditions. First, we discuss how exactly we incorporate boundary conditions into the prediction of surrogate models. Then we discuss how to handle pixels whose associated nodes lie outside the geometry associated with the pixel image.

The solution of a BVP strongly depends on the chosen boundary conditions. Therefore it is essential to integrate them into our physics-aware approach, as we train the model entirely without reference data. There are at least two ways to enforce the boundary conditions in the context of a CNN. One possibility is to extend the loss function by an additional loss term associated with the boundary conditions, for example $\|B(X)\|_2^2$. This is referred to as *soft enforcement* in the literature [176]. However, this introduces an unintended trade-off between satisfying the physics constraints, i.e. the momentum and mass equations, and the boundary conditions. Additionally, there is the question of the weight we assign to the additional loss term. It has been shown that soft enforcement of constraints can be problematic, e.g., not satisfying the boundary conditions while minimizing the residual can lead to wrong predictions [176].
The other possibility is to include the boundary conditions directly in our network and thus always satisfy them exactly. The latter approach corresponds to hard-coding the boundary conditions and is closer to the conventional implementation of Dirichlet boundary conditions, e.g. in the finite difference or finite element method, and it is referred to as *hard enforcement*.

In our approach, we implement the Dirichlet boundary conditions for u, v, and p directly in the network, i.e., hard enforcement. To accomplish this, we overwrite the values of u_{NN}, v_{NN}, and p_{NN} in the relevant pixels with the correct values after the prediction is complete but before evaluating the residuals. In this way, we guarantee that our models exactly satisfy the boundary conditions in the pixels on the boundary. Consequently, we do not enforce the physics constraint in the respective pixels.

Using the example of two-dimensional channel geometry, see section 2.3, this entails setting u and v on the inlet

$$u_{i,j} = 3, \quad v_{i,j} = 0,$$

u and v on the no-slip boundary

$$u_{i,j} = 0, \quad v_{i,j} = 0,$$

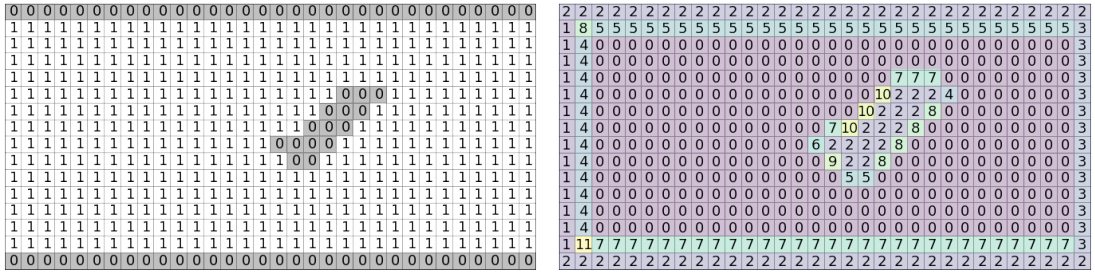
and p on the outlet

$$p_{i,j} = 0.$$

This process is straight-forward for Dirichlet boundary conditions. Neumann boundary conditions, on the other hand, cannot be set in the same way. Here, however, we can again draw inspiration from the procedure in the finite difference method. There, so-called ghost nodes are introduced to fulfill the Neumann boundary conditions. Such a procedure is also possible in the context of CNNs. However, since we do not consider Neumann boundary conditions, we refer to e.g. [176]. Note that implementing Neumann boundary conditions with our approach can present additional difficulties. Often, Neumann boundary conditions consist of specifications for derivatives in the normal direction with respect to the boundary orientation. However, our approach is not designed for a single geometry with fixed boundaries and the orientation of the boundary may vary from geometry to geometry. Therefore, this orientation must be known or approximated at the time the boundary conditions are applied. Further, the orientation may need to be determined on a pixel-by-pixel basis for jagged boundaries, such as the obstacle boundaries in the pixel images of the channel geometries; see section 4.4. Finally, the rasterized boundaries created by interpolating to a pixel image can introduce corner points on the boundary where the normal vector is not well defined.

To identify which boundary conditions have to be set at which pixel, we extend the input of our physics-aware model by an additional pixel image of the geometry. This additional pixel image contains boundary condition treatment information for each pixel.

In the following, we refer to this additional pixel image as *boundary image*. An example of a geometry pixel image and the corresponding boundary pixel image pair is shown in fig. 5.3. Let us note that the boundary image is not an additional input to the underlying CNN but is solely used to enforce the boundary conditions and to identify where the residual can be computed. In the boundary image each pixel is assigned an integer number



(a) Geometry image (b) Boundary image

Figure 5.3: Low-resolution pixel image inputs that are used by our model. The geometry image (a) is passed as input to the CNN and the boundary image (b) is used for the construction of the physics-aware loss.

that identifies whether the node associated with the pixel is located in Ω, on one of the edges of Ω, or outside Ω. Further numbers indicate in which direction from this pixel an edge pixel is located, where the pressure is not defined. There we use one-sided differences to approximate the corresponding derivative of the pressure.

In detail, the number 0 corresponds to internal nodes that do not require any special treatment. The numbers 1, 2, and 3 denote nodes that lie on the inflow, no-slip, and outflow boundary, respectively. Here we prescribe boundary conditions. The numbers from 4 indicate nodes where we must use one-sided approximations for the pressure gradient to avoid using pressure values at pixels where the pressure is not defined. Note that there are pixels whose associated node is neither in the Ω geometry nor on the boundary. In such cases, the values for both velocity and pressure are not defined. We

also indicate such pixels with a 2 in the boundary image, so that they are treated as if they were on the no-slip boundary. Consequently, the velocity is set to 0 and, most importantly, the physics constraint are not enforced. In practice, the local residuals at these pixels are set to 0. Thus, the value of the residuals at these pixels have no effect on the loss, since we fit the residuals at each pixel to 0.

5.5 Physics-Aware Model

The description of a physics-aware CNN has been rather theoretical so far. In this section, we explicitly outline the steps necessary to extend a standard CNN to a physics-aware CNN.

We assume that we have a CNN that takes a pixel image of the geometry as input and returns pixel images of the solution variables as output, for example as defined in section 4.5. Note that the input can actually take other forms, the important part of the underlying CNN is the output. To extend this CNN to a physics-aware CNN, we need to implement the treatment of the boundary conditions, the computation of the partial derivatives, and the computation of the residuals with regards to the Navier–Stokes equations. A visualization of this process is shown in fig. 5.4.

We discussed the handling of boundary conditions in section 5.4. For this we need to extend the input with an additional pixel image, the boundary image. This image assigns an identifier to each pixel in the form of a number; see fig. 5.3. We use this identifier to explicitly set the boundary conditions, to determine in which pixels we need to apply one-sided approximations to compute the derivatives, and to enforce physical conditions only where they are valid. In practice, this can be easily achieved by utilizing the boundary image as a mask for the respective pixel images.

We have outlined the general process of how we approximate partial derivatives for a pixel image using convolutions in section 5.1. In practice, for each partial derivative, we apply a convolution with fixed weights to the pixel image of the solution variable with the

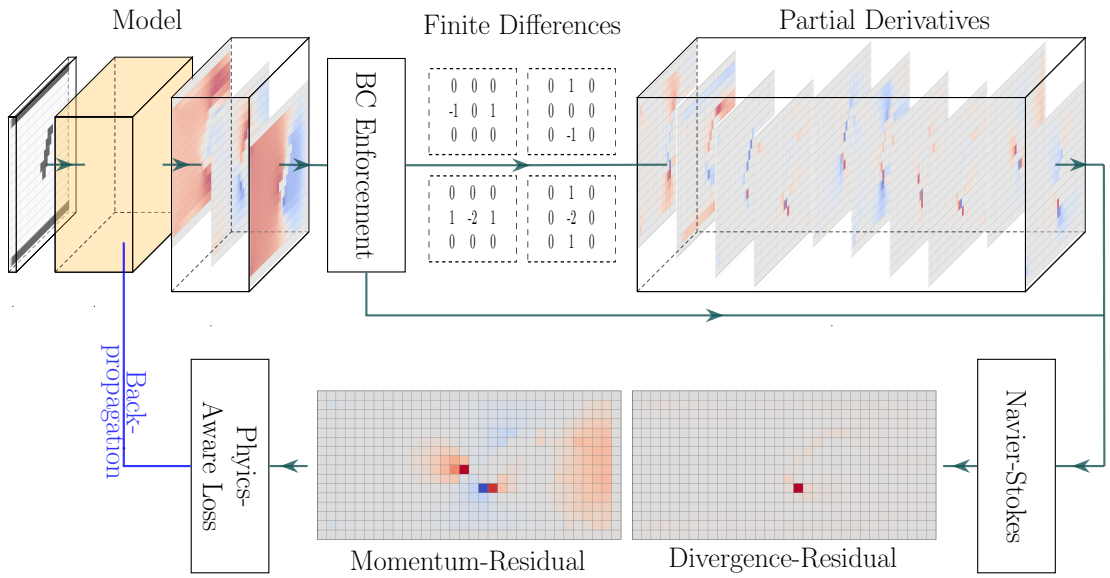


Figure 5.4: Physics-aware convolutional neural network for the Navier–Stokes equations.

boundary conditions already enforced, so that the filter corresponds to a finite difference stencil. This gives us an approximation of the partial derivatives in each pixel.

To calculate the residuals we perform the corresponding operations, i.e., multiplication and addition, see eq. (5.18) - (5.20), pixel by pixel. Most ML frameworks enable us to do this directly on the image level. For this we use the pixel images of the solution variables with the boundary conditions enforced from the first step and the pixel images of the partial derivatives from the second step. In a last step, we again employ the boundary image as a mask to set the computed residuals to zero in pixels that are outside of the geometry.

A CNN extended in this manner takes two separate pixel images as input and returns six pixel images as output. In our case, the input images are a pixel images of the geometry and boundary images that contains information about the boundaries. The six output images comprise one each for the three solution variables u, v, and p, one each for the residuals of the x- and y- components of the momentum equation, and one each

for the residual of the mass equation. Using the pixel images of the residuals, we can construct a physics-aware loss, see eq. (5.26), to train our model to discretely satisfy the Navier–Stokes equations.

6 Results in Two Dimensions

In this chapter, we explore and analyze our method using two-dimensional cases. We consider the model problems introduced in section 2.3. Given the multitude and diversity of hyperparameters and choices regarding the tools to approximate the derivatives needed in the calculation of our loss, we examine several variants of our method in this section. For this purpose, in the following we mostly first present and discuss some results on single geometries for each investigated variant and then apply this variant to multiple geometries. However, due to time constraints, we do not apply each variant to multiple geometries, as training on several thousand geometries would consume an exorbitant amount of time and computational resources.

In general, we can split the error of the prediction of our model into different parts: e.g., the truncation error due to the Taylor approximations; the discretization error due to the division of the domain by a mesh; the misrepresentation error due to the representation of the geometries on a uniform cartesian mesh; the training error, i.e., how well we solved the optimization problem; the generalization error when applying our model to new geometries not seen in training; and the interpolation error that exists in our reference data due to the interpolation of the results of an FV simulation onto our pixel grid. When training on a single geometry, the generalization error is eliminated and the training error should be lower, since the optimization problem on one geometry is most likely less complex than on multiple geometries. At the same time, training in this case is less computationally expensive, and thus we can train more models in the same amount of

time. Of course, when applied to a single geometry, our model no longer corresponds to a reasonable surrogate model, but we still consider this case in order to study our method under such circumstances.

Problems with our method can occur due to several different reasons. However, we can roughly separate such problems into two different categories – general machine learning problems; cf. section 3.4, and finite difference problems, in general and specifically with respect to the Navier-Stokes equations; cf. section 2.2.3.

In this chapter, we apply the physics-aware approach in different scenarios and analyze its performance. First, in section 6.1, we examine the effect of some hyperparameters such as the chosen resolution and model complexity as well as the impact of random initialization of the networks parameters on individual geometries of the channel data set. Then, in section 6.2, we apply the base-variant of our method presented in chapter 5 first to single channel geometries in section 6.2.1 and single artery-geometries in section 6.2.2, and subsequently to multiple channel geometries in section 6.2.3 and multiple artery geometries in section 6.2.4. In section 6.2.3, we analyze the performance of the physics-aware method in detail and compare it in particular with the state-of-the art data-based approach. We also investigate a combination of the two methods there. After that, we show in section 6.3 that our method is able to handle even more variations such as boundary conditions by slightly changing the models used. Finally, in section 6.4, we discuss some modifications to the physics-aware approach and analyze their impact on the performance of our models. Some of the results shown here were previously presented in [55].

All models presented in this chapter were implemented in TensorFlow v. 2.5, 2.7 or 2.9 [1]. We always use the Adam optimizer [85] to train our models.

6.1 Some Preparatory Comments

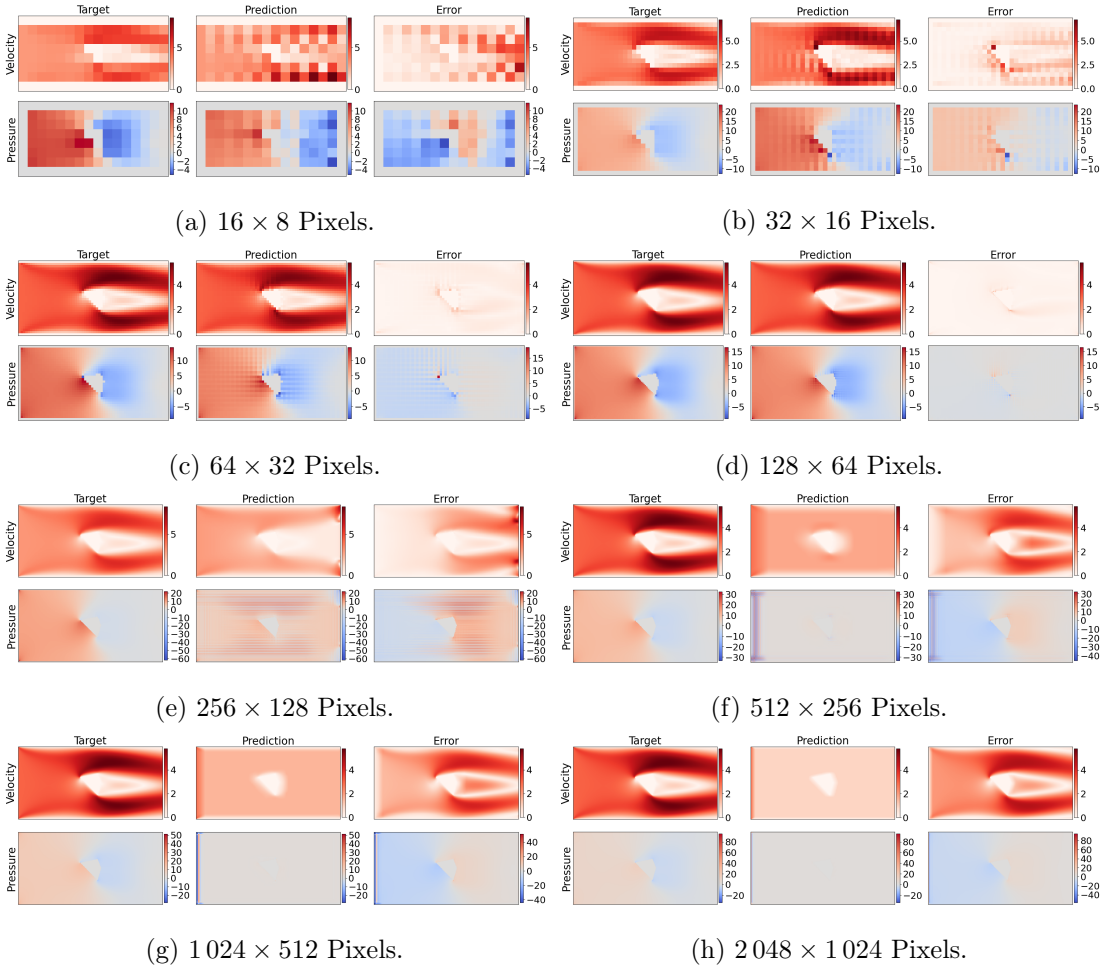
In this section, we discuss some general machine learning aspects of our approach. First, in section 6.1.1 we discuss the resolution of the pixel images we use, as well as the chosen hyperparameters, especially the depth of the CNN. Then in section 6.1.2 we discuss the consistency of the training, i.e. how reliably we are able to find a suitable minimum of the loss function.

6.1.1 Choice of Resolution

For this work we always consider images with a resolution of 256×128. There are many reasons for this, some of which we discuss in this section. One of them is because this is the resolution chosen in the relevant literature [35, 36, 58]. Another is because with this resolution in the multiple geometry case, we have data sets that are still of a manageable size. This is especially important in the three-dimensional case, where data sets are already 10 or 20 gigabytes in size for just a few geometries; cf. section 7.1. That is why, in this section, we qualitatively study the effectiveness of our model for a fixed geometry in different resolutions. In all models studied here, ReLU is the activation function, the learning rate is 10^{-4} and centered differences of second order were used.

The choice of our network architecture limits us to resolutions of the form $2^{nl+1} \times 2^{nl}$, where nl is the number of levels of our model. However, this is not a general limitation. For example, we could use a slightly modified form of our architecture to allow us to process resolutions of the form $3^{nl+1} \times 3^{nl}$. With suitable padding for the down- and upsampling layers as well as skip connections, it is also possible to use much more arbitrary resolutions.

We consider 16×8 as the smallest resolution and $2\,048 \times 1\,024$ as the largest resolution. Accordingly, the deepest model we can train on all resolutions consists of 4 layers. So, as a first step, we train a model with 32 start channels and 4 levels on all chosen resolutions. The predictions of the respective models after 500 000 epochs of training are shown in



(a) 16×8 Pixels.

(b) 32×16 Pixels.

(c) 64×32 Pixels.

(d) 128×64 Pixels.

(e) 256×128 Pixels.

(f) 512×256 Pixels.

(g) $1\,024 \times 512$ Pixels.

(h) $2\,048 \times 1\,024$ Pixels.

Figure 6.1: Predictions of models that are 4 levels deep for various resolutions.

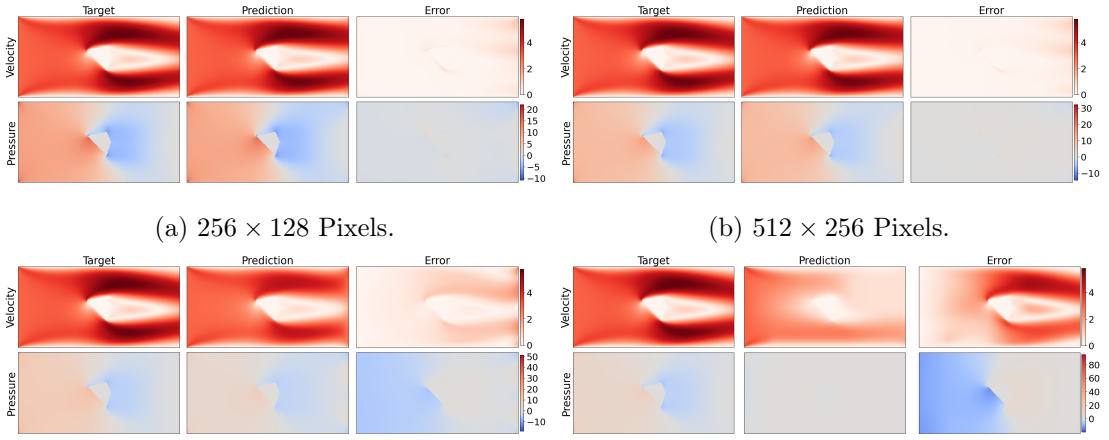
fig. 6.1. Here we can immediately observe several effects, some of which we have already addressed in section 2.2. Let us note that 500 000 epochs are more than is necessary; however, with such a long training duration we ensure that our model is fully trained.

First, we see oscillations in the velocity field of the 16×8 and 32×16 predictions. This is most likely due to the fact that we have approximated the convective term with central FD stencils. If the grid is too coarse, this can lead to oscillations in the velocity; see Cell-RE problem in section 2.2. There are means to overcome this problem. However, these

resolutions are so coarse that the features of the geometry are not resolved sufficiently. This so drastically falsifies the learned predictions that we do not continue to look into small resolutions.

Second, we see that the predictions of the 256×128 and higher resolution are far from the reference solution. In fact, in the 256×128 prediction, strong oscillations are present in the pressure. In the predictions of the higher resolutions, these oscillations are limited to a region near the inflow, but apart from that the pressure is constant 0 here. There are many possible explanations for the occurrence of these oscillations. One is strongly connected to the inf-sup stability condition, respectively the pressure-velocity coupling. Another is the cell RE-problem; cf. section 2.2.3. Possible solutions to this include utilizing regularization. Another is related to the architecture used. Our models used here all consist of only 4 levels. At the higher resolutions, this implies that no information from the left part of the geometry image is used in the calculation of the velocity value of a pixel at the far right edge. For example, at a resolution of 256×128, the activations in the deepest layer are 16×8. A 3×3 convolution is then applied to this. In the subsequent decoder-part, the two peripheral areas of the geometry move further away from each other again. Thus, one possible solution is to train deeper models at higher resolutions. However, there are many other possible causes of this problem, several of which are typical ML problems and not exclusively specific to this application. We have addressed some of them in section 3.4.

On higher resolutions, we thus train models that are 8 levels deep. The corresponding predictions are shown in fig. 6.2. We can clearly see that the learned predictions have improved with the additional depth of the models. The predictions for the resolutions 256×128 and 512×256 seem convincing, as the remaining errors are limited to the immediate surrounding of the obstacle and the closer vicinity of the outflow. One is caused by the rasterized depiction of the geometry in our input images; cf. section 4.4, and the other by small oscillations in the pressure close to the outflow boundary. We



(a) 256 × 128 Pixels.

(b) 512 × 256 Pixels.

(c) 1 024 × 512 Pixels.

(d) 2 048 × 1 024 Pixels.

Figure 6.2: Predictions of models that are 8 levels deep for various larger resolutions.

address both of these circumstances later. Note that the outflow boundary condition is not a natural boundary condition, but is dictated by the necessity to clip off a domain for computational reasons; cf. [53]. Therefore, the boundary condition we impose at the outflow boundaries may not be the best option. We refrain from discussing this problem in detail and refer to the literature, e.g., [15, 40, 53, 101, 105, 135, 148, 188, 195].

For now let us focus on the predictions for the resolutions 1024 × 512 and 2048 × 1024. Both prediction are already much closer to the reference solution than the previous predictions made by models with 4 levels, but not close enough. We can still observe strong differences. In order to be able to obtain meaningful predictions at the two higher resolutions, we can consider several options. For example, we could train models with even more levels. However, an even deeper model will have even more parameters, i.e. we would need more memory and more computational resources to train such a model. These factors are not yet major problems in the two-dimensional case, but they are in the three-dimensional one; cf. section 7.1. However, at this point it should be noted that training the models for multiple geometries is very time-consuming even in the two-dimensional case. At the same time, it is difficult to successfully train deeper and

deeper models for various reasons, e.g. vanishing gradients; cf. section 3.4. Here, the skip connections in our architecture do not help with respect to gradients in the deepest layers. We should therefore generally try to keep the model architecture at a reasonable size. Also, further testing in this area has shown that we gain only limited improvement with even deeper model architectures without additional adjustments to the loss calculation and/or training process. We can, however, try other techniques such as using a different learning rate schedule or introducing pressure regularization. We cover some of those modifications in section 6.4.

6.1.2 Consistency of Training

When training a neural network, certain problems frequently occur; cf. section 3.4. One of them is that due to the highly nonconvex nature of the loss function, training a neural network can lead to very different results depending on the initialization of the model weights. Given an identical configuration, a different result might be obtained each time. In general, this is a undesirable property. In this section we investigate just how different the obtained results can be on the example of a single channel geometry. Again, in all models studied here, ReLU is the activation function, the learning rate is 10^{-4} and centered differences of second order were used. We train four models that are 8 levels deep and have 32 start channels for 500 000 epochs and compare the respective results.

We refer to the four different models as first, second, third and fourth run. The relative errors of the predictions are, in respective order, 6.53%, 5.76%, 3.31%, and 2.97% in u and 20.76%, 17.90%, 8.53%, and 8.86% in p. The corresponding predictions are shown in fig. 6.3. Thus, although all the models are identical except for the initialization of the parameters and have been trained identically, the learned predictions are different. The errors vary by as much as 4% for velocity and 12% for pressure. We will see later in this chapter that these variations can sometimes be even larger; see, for example, section 6.4. This variation is quite large. Thus, if we are interested in the best model, we are forced

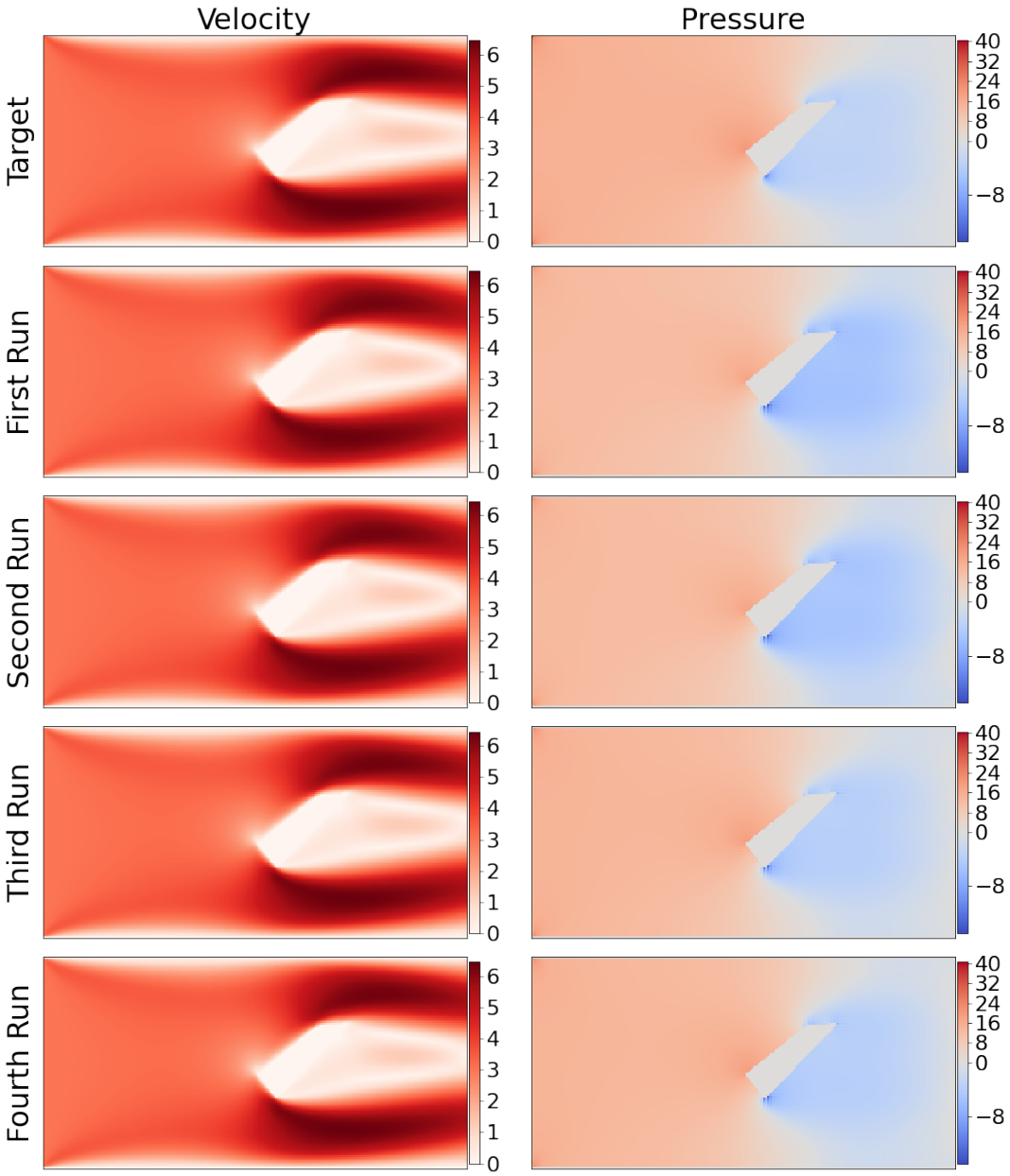


Figure 6.3: Predictions of models with the same architecture but with different initializations.

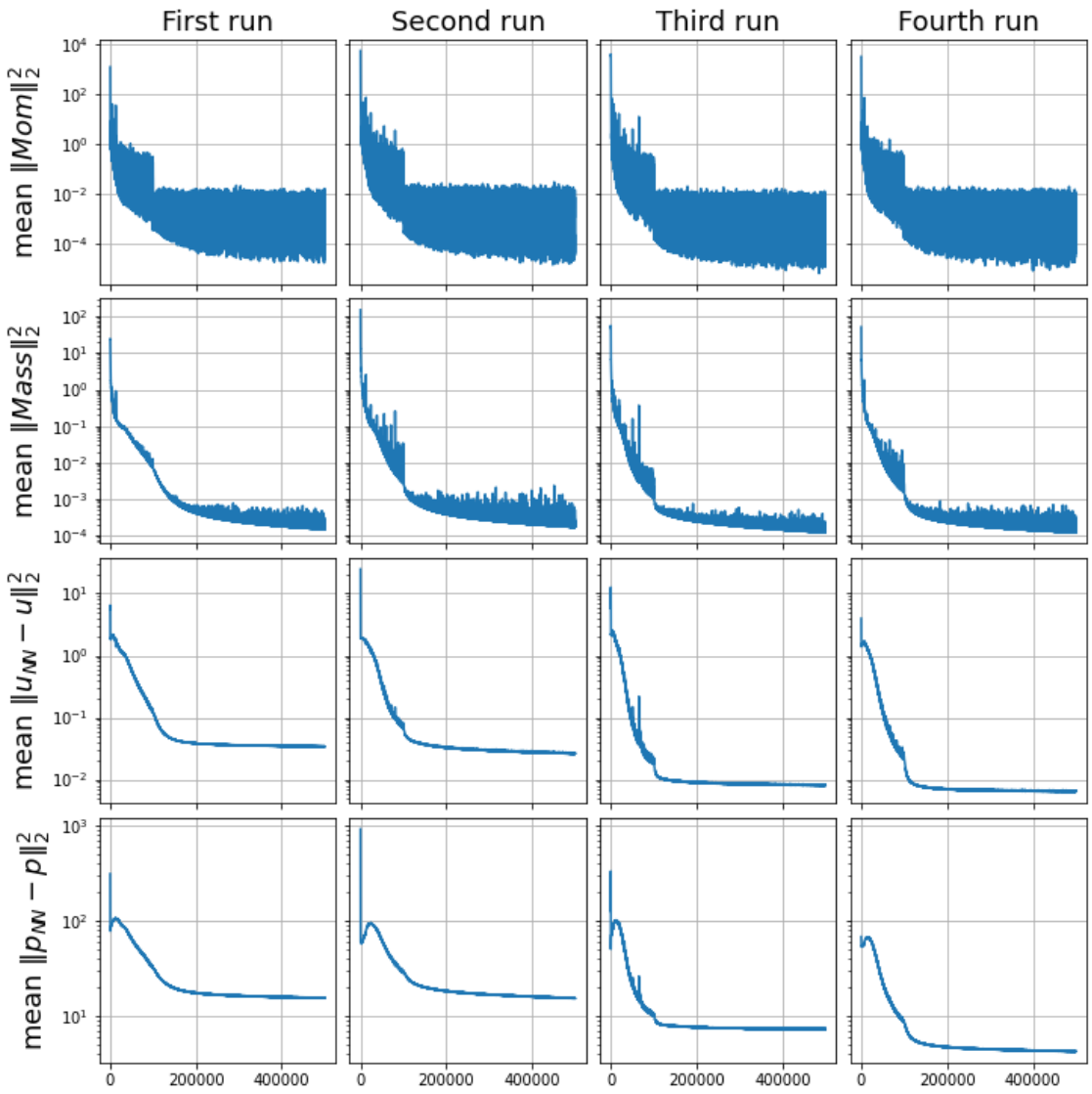


Figure 6.4: Loss history of models that have the same architecture but with different initializations.

to train multiple models and compare their performance. This complicates the study of our approach.

In fig. 6.4, the history of the physics-aware and data-based loss terms during training

is shown. Note that only the physics-aware loss terms were used to train the models, and the data-based loss terms were used only for evaluation. It is interesting to note, that the general behavior of the physics-aware loss-terms is very similar over the course of training. In general, the values of the mean squared residuals are quite volatile. However, the trend is slowly decreasing. After about 200 000 or 250 000 epochs, there is only little improvement gained. There are small differences, especially in the final values of the residuals. Thus, the value of the mean squared momentum residual for the individual runs is $3.9 \cdot 10^{-4}$, $4.5 \cdot 10^{-4}$, $7.9 \cdot 10^{-5}$, and $1.6 \cdot 10^{-3}$, and the value of the mean squared mass residuals $1.5 \cdot 10^{-4}$, $1.8 \cdot 10^{-4}$, $1.2 \cdot 10^{-4}$, and $1.3 \cdot 10^{-4}$. No reliable trend with regards to the learned predictions can be derived from these values.

The results presented in this section show that training a physics-aware CNN yields different results depending on the initialization of the parameters, even if all other parameters and hyperparameters remain the same. This behavior is typical for deep neural networks; see section 3.4. Therefore, in the rest of this thesis, either the predictions of multiple models are considered, or only the prediction of the best model obtained is presented and examined.

6.2 Proof of Concept

In this section, we apply the base-variant of our physics-aware convolutional neural network method to the geometries defined in section 2.3. By base-variant we designate a physics-aware CNN wherein we use only centered differences of second order to compute the physics residuals and do not make any additional modifications. In particular, this means that the loss terms are all equally weighted. Here we examine the capabilities and shortcomings of our approach.

To do this, we first consider the application of our method to single channel geometries in section 6.2.1. This allows us to examine our method without the additional source of error of generalization. This depends on more than just our method, e.g. in particular the

design of the training data set and the validation data set. We then apply our method to single artery geometries in section 6.2.2 to evaluate how our model handles non-rectangular geometries as well as realistic boundary conditions and material parameters. Finally, we examine our method for multiple channel geometries in section 6.2.3 and multiple artery geometries in section 6.2.4.

6.2.1 Single Channel Geometries

In this section we test and analyze the capabilities and shortcomings of our approach on various single channel geometries; cf. section 2.3.1.

Training on a single geometry is a rather large simplification. Indeed, the function to be approximated is much simpler for only one geometry than for multiple geometries. It is to be expected that the loss landscape is significantly less non-convex than in the multiple geometry case. At the same time, we eliminate the generalization error and can effectively overfit very strongly to a single geometry. This allows us to train our model on a particular geometry to be as accurate as possible. That is, we expect the learned prediction for that geometry to be an upper bound on performance for our approach, since training on multiple geometries involves additional sources of error and is thus very likely produce worse results.

We show results for three geometries with widely varying obstacles, leading to velocity fields with different maximum velocities ranging from $6\frac{m}{s}$ to $13\frac{m}{s}$. We have trained a variety of models with different hyperparameters on these geometries, but here we show only the best results obtained. The purpose of this section is not to show how to optimize hyperparameters for our method or to present the results of a grid search, but to show what our approach is capable of and to illustrate limitations. All models for which we present results here were trained for 250 000 epochs.

6.2.1.1 First Geometry - Maximum Velocity $6\frac{m}{s}$

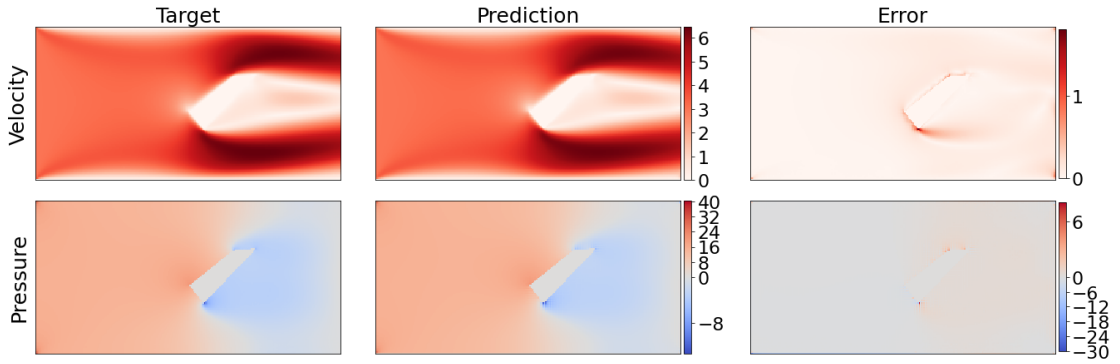
The obstacle of the first geometry is four-sided and covers about one third of the channel in height. It is placed vertically approximately in the center so that there is equal distance to the boundaries at the top and bottom. The reference solution has a maximum velocity of $6.5\frac{m}{s}$, which occurs at the narrowed areas caused by the obstacle. We obtained the best result on this geometry using a model with swish as the activation function and a learning rate of $5 \cdot 10^{-5}$. The corresponding predictions and the error compared to the reference solution is shown in fig. 6.5(a).

Our physical model is able to learn both the flow field and the pressure very well. The relative L_2 error for u is only 2.6% and 2.8% for p. However, the learned flow field deviates slightly from the simulated flow field, especially in the proximity of and behind the obstacle. There are several possible reasons for this.

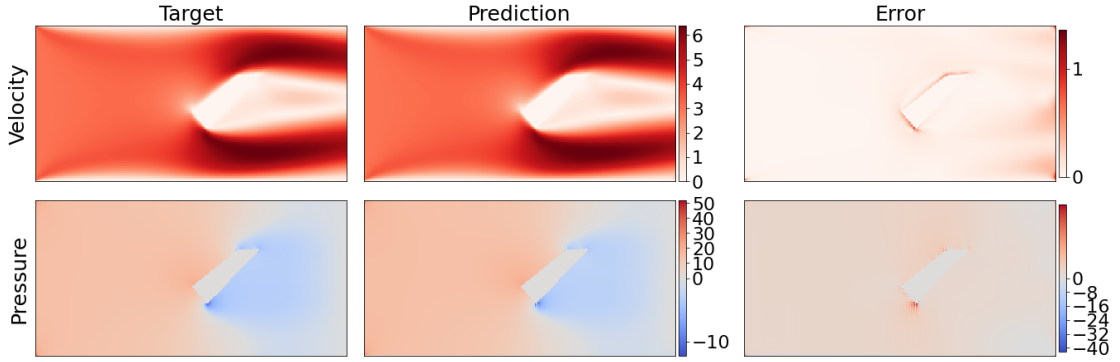
First, we know the image of the geometry is inaccurate since it is represented as a pixel image; cf. section 4.4. Here, this refers to the representation of the obstacle, where a rasterization of the boundaries occurs. In the original representation the boundaries of the obstacle are straight, but in the pixel image they are step-like.

Second, the resolution with which our model works is significantly lower than the resolution with which the reference solution was computed. At a resolution of 256×128, the images, and thus the mesh our method works with, consist of about $32,000$ nodes (or pixels), not all of which are in the computational domain. For comparison, the reference solution was computed with a mesh consisting of about 180 000 elements. Moreover, this mesh is not uniform, unlike the mesh of our method, but adapted to the expected behavior of the fluid.

Thirdly, the considered reference solution can be regarded as ground truth, but it does not correspond to an analytical solution and is not entirely devoid of errors. It is the result of an FV simulation that was solved iteratively until a relative tolerance was reached; cf. section 4.4.



(a) Velocity and pressure for the physics-aware approach (Prediction) compared to the OpenFOAM simulation on a *locally refined* mesh (Target).



(b) Velocity and pressure for the physics-aware approach (Prediction) compared to the OpenFOAM simulation on a *cartesian rasterized* mesh (Target).

Figure 6.5: Results for the first geometry.

To address two of the aforementioned factors, we propose the creation of a mesh that encompasses the geometry of interest while incorporating the same erroneous representation of the obstacle and comparable resolution. To achieve this, this mesh consists of squares exactly equal to the size of a pixel of the pixel images our model operates on, and we refer to it as a Cartesian rasterized mesh. Any square whose center is not in the geometry will not be part of the mesh. So we get a mesh with the same incorrect representation of the geometry. This mesh can then be used for simulation. Notably,

since the physics-aware approach we use employs a finite difference approach, while the simulation method we use utilizes the finite volume method, generating a mesh that matches the pixel image from the physics-aware approach is not possible. Thus, there will be residual errors, especially near the obstacle boundary. To create such a Cartesian rasterized mesh with the obstacle, we have chosen to cut out pixels from the channel where the centers lie within the obstacle. Such a Cartesian rasterized mesh based on an image of lower resolution can be seen in fig. 6.6. Note that we have halved the pixels on the outer boundaries of the channel to ensure enforcement of the correct boundary conditions.

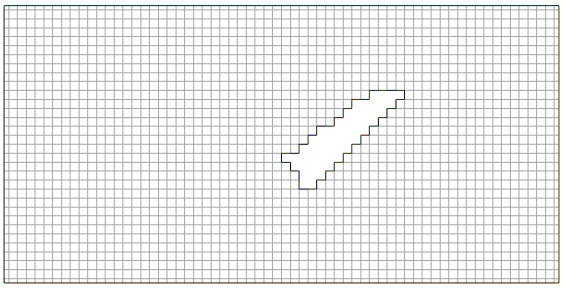


Figure 6.6: Cartesian Rasterized mesh of the first geometry, here based on a lower resolution image of 64×32 pixels.

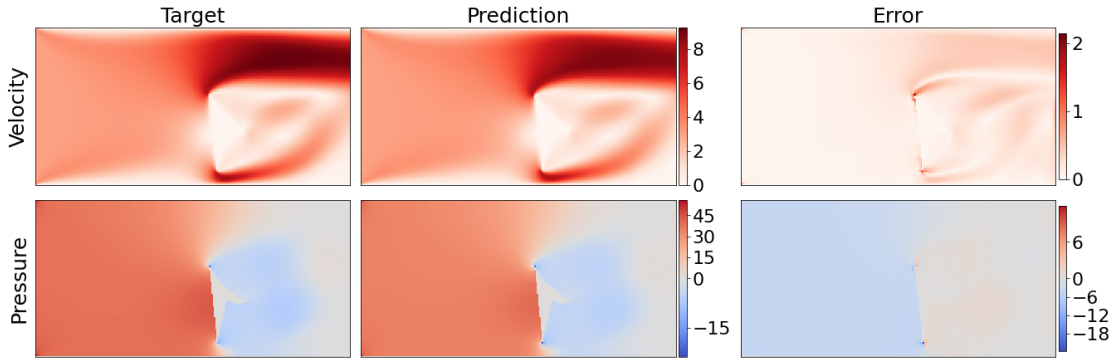
The result of the simulation on the Cartesian rasterized mesh is shown in comparison to the prediction of our model in fig. 6.5(b). Here the relative L_2 error in velocity has decreased to 2.2% (from 2.6% before). In particular, the error has decreased in the immediate vicinity of the obstacle; note the change in the scaling of the colorbar of the error plot here. However, the error in the pressure has increased to 4.9% (2.8% before), with the greatest error being at the southernmost tip of the obstacle. This difference can most likely be explained by the use of an FD scheme on a co-located grid in our method compared to a FV scheme on an, at least partially, staggered grid in the reference simulation. This leads to a different pressure, and the effect is larger for the low resolution Cartesian rasterized meshes.

6.2.1.2 Second Geometry - Maximum Velocity $9\frac{m}{s}$

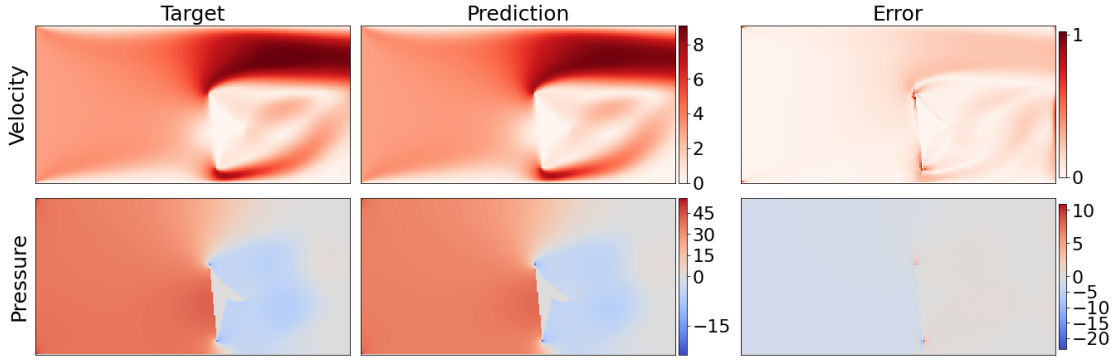
The obstacle of the second geometry is four-sided and covers about half of the channel in height. It is located more in the lower half of the channel, leaving little space below the obstacle to the lower boundary of the channel. Accordingly, the reference solution has a maximum velocity of about $9\frac{m}{s}$, assuming this value above the obstacle. This results in an interesting flow behavior, which in some areas is more strongly characterized by convection than in the previously studied geometry. Here, we obtained the best results with swish as the activation function and a learning rate of $1 \cdot 10^{-4}$. The corresponding predictions and the error compared to the reference solution is shown in fig. 6.7(a).

The prediction of our model is again quite close to the reference solution. The relative L_2 errors in velocity and pressure are 4.7%, and 10.0%, respectively. However, these errors are already noticeably higher than in the case of the previous geometry. The biggest difference from the previous geometry is the higher maximum velocity, and thus the stronger convection. In fact, we see that the error is again mostly high in the immediate vicinity of the obstacle and in the area directly behind and above it. In contrast to the previous geometry, however, the error in the area behind and above the obstacle has become larger. This is exactly the area where the fluid flows faster than before and thus the flow behavior is, in principle, more strongly dominated by convection. However, we know that in FD schemes with convection-dominated flow, the convective terms of the associated PDE should not be approximated with central stencils; cf. section 2.2. Since we exclusively use central stencils of second order for the calculation of the residuals in the model trained here, it is therefore not surprising that additional diffusion occurs in the part above and behind the obstacle and thus also higher deviations from the reference solution.

As before, we can create a Cartesian rasterized mesh of this geometry. We can then compute a simulation on it and compare the result with our prediction. This comparison is shown in fig. 6.7(b). Compared with the simulation on the Cartesian rasterized mesh,



(a) Velocity and pressure for the physics-aware approach (Prediction) compared to the OpenFOAM simulation on a *locally refined* mesh (Target).



(b) Velocity and pressure for the physics-aware approach (Prediction) compared to the OpenFOAM simulation on a *Cartesian rasterized* mesh (Target).

Figure 6.7: Results for the second geometry.

which features the same erroneous representation of the obstacle and is of similarly low resolution, the relative L_2 errors in velocity and pressure decrease to 2.7% and 5.1%, respectively. This means that in this case the low resolution and the incorrect representation of the obstacle has a greater impact on the flow behavior than before.

Furthermore, in the case of a faster flowing fluid, larger derivatives can be expected if the geometry remains the same. With this geometry, the obstacle covers more of the channel than with the previous one, which means that even greater derivatives are to be

expected. The truncation error we introduce by using second-order FD stencils might in this case become too large, so it might be sensible to use higher order FD stencils.

6.2.1.3 Third Geometry - Maximum Velocity $13\frac{m}{s}$

The obstacle of the third and last geometry we inspect in this part is a large thin triangle that covers two thirds of the channel in height. It is positioned vertically in the center. This obstacle blocks a large part of the channel and leads to a flow field with a maximum velocity of almost $13\frac{m}{s}$. Consequently, the flow at the bottlenecks caused by the obstacle and behind it is dominated by convection even more than in the two previous geometries. Here, we obtained the best results with swish as the activation function and a learning rate of $2.5 \cdot 10^{-4}$. The prediction of the best model and the error compared to the reference solution is shown in fig. 6.8.

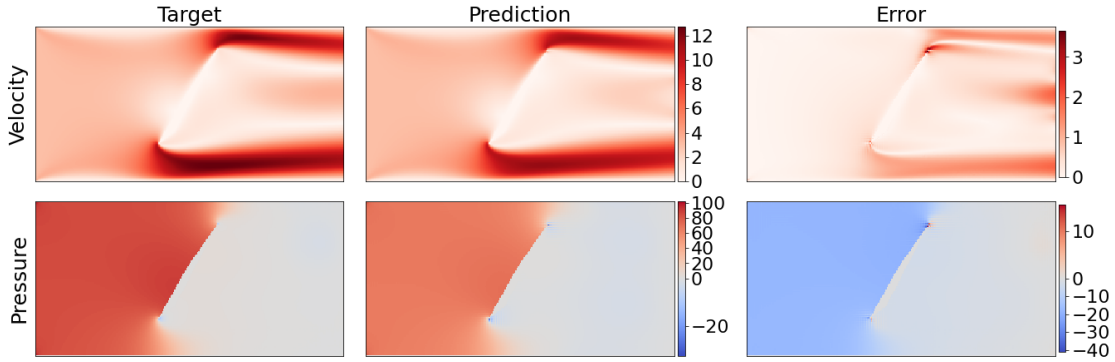
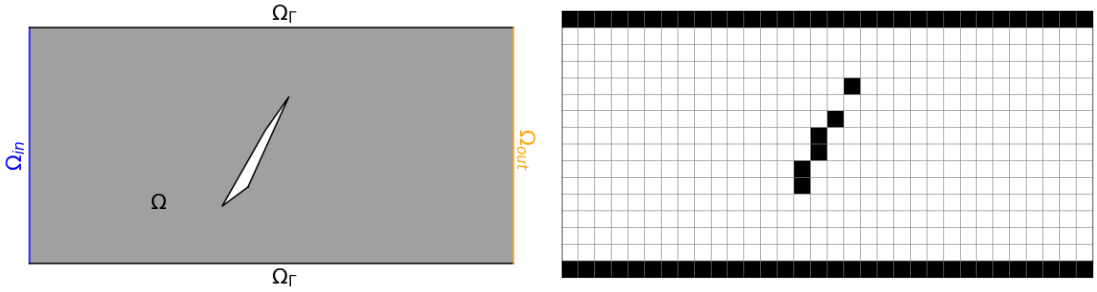


Figure 6.8: Results for the third geometry. Velocity and pressure for the physics-aware approach (Prediction) compared to the OpenFOAM simulation on a *locally refined* mesh (Target).

It is clearly visible that our physics-aware model has difficulties in learning a correct flow field. The relative L_2-errors are 11.2% for the velocity and 22.3% for the pressure. The error is again high in areas where the flow is more dominated by convection rather than diffusion. The maximum of the predicted speed is about 10% lower than the

actual maximum. Fittingly, the difference in predicted pressure upstream of the obstacle compared to downstream of the obstacle is less than the pressure of the reference solution. However, these issues cannot all be attributed to the low order of the FD stencils used and the low resolution. An important reason, which should not be neglected, is that the obstacle, which is actually solid throughout, was separated into two parts at the top due to the incorrect representation. In the prediction of our model, a non-negligible part of the fluid flows through this gap in the obstacle and thus reduces the pressure, like a kind of valve. See fig. 6.9 for a low-resolution representation of this gap in the obstacle. This gap does not disappear even with finer resolutions, it just moves further up.



(a) (b)

Figure 6.9: Real geometry (a) and 32×16 pixel image representation (b).

We could, as before, try to explain part of the error by comparing our prediction to a simulation on a mesh with the same erroneous representation of the geometry (including the obstacle) – and with a similar level of fine-grained detail. However, such a simulation does not converge for this geometry. Here, the flow oscillates behind the obstacle and does not converge to a stationary solution. Combined with the fact that the simulation converged on a finer mesh, this suggests that the resolution of the mesh used is insufficient for the boundary value problem defined by this geometry. We should therefore also not be surprised that the training of physics-aware models on this geometry is extremely volatile. Even if we train models that are exactly the same except for the initialization of

the weights, the predictions of the final trained models can be radically different.

In summary, in this section we have tested the physics-aware approach on three very different channel geometries. We found that our model is basically able to learn correct solutions for the first two geometries shown. However, we observed that our model has increasing difficulty for geometries that cause higher velocities in the velocity field.

6.2.2 Single Artery Geometry

In this section we demonstrate the capability of our approach to work with non-rectangular geometries. So far we have only considered channel geometries, which are in particular rectangular geometries. We described how to handle geometries that are not rectangular in chapter 5, and how to create suitable images for them in section 4.4. In the following, we explore this experimentally and investigate difficulties encountered.

As a test case for a non-rectangular geometry, we consider a bifurcation geometry that was generally described in section 2.3.3.1. The geometry considered here consists of arteries that are uniformly 0.004m wide. Fluid flows in from the left in a single artery and then splits into two arteries branching upward and downward at 45° each. The geometry also features a small saccular aneurysm at the right boundary of the lower artery.

We obtained the result presented here using a model with swish as the activation function and a learning rate of $5 \cdot 10^{-5}$. The corresponding predictions and the error compared to the reference solution is shown in fig. 6.10. The prediction generally fits very well to the reference solution. However there is a larger deviation towards the lower outflow and another smaller one towards the upper outflow. Consequently, the relative L_2-errors are 9.8% for the velocity and 34.65% for the pressure.

In general, we can conclude from this result that our approach is able to handle non-rectangular geometries. For a non-rectangular geometry, however, the boundaries are rasterized, as was previously the case for the obstacles with the channel geometries. The difference here is that now almost all boundaries are rasterized, so there are almost no

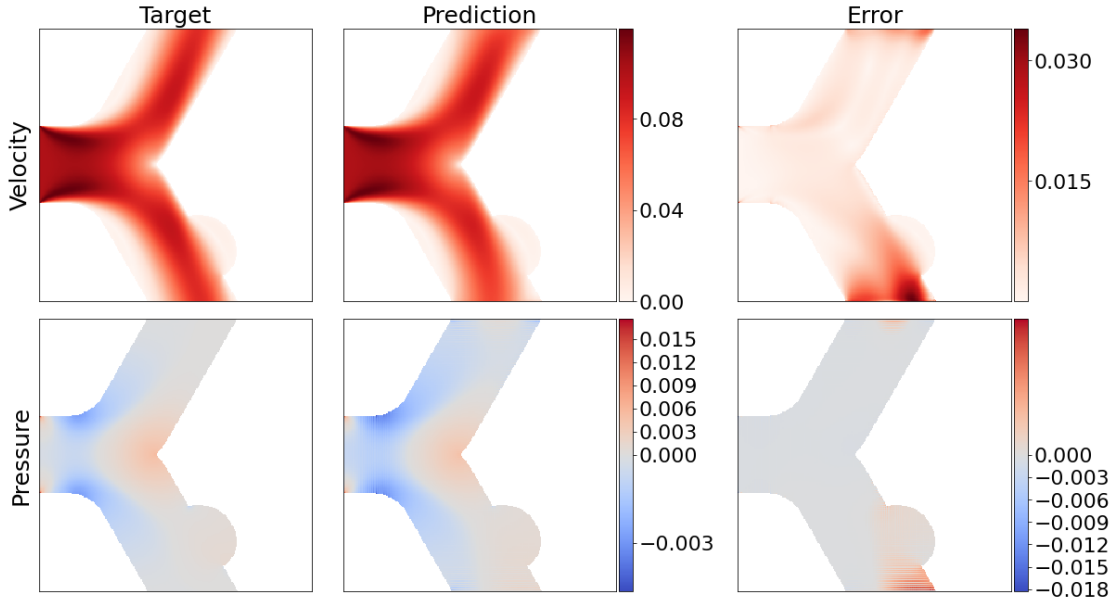


Figure 6.10: Velocity and pressure for the physics-aware approach (Prediction) compared to the OpenFOAM simulation on the *locally refined* mesh (Target). The model was trained on a single geometry.

straight boundaries except for the in- and outflow boundaries. This effect, in combination with the use of centered differences, here of second order, favors the occurrence of checkerboard oscillations in the pressure. In fact, there are significant oscillations in the pressure near the inflow and outflow boundaries; see fig. 6.11 for a zoom into the areas close to the boundary. Especially near the outflow boundaries, these oscillations influence and falsify the predicted flow field.

The problem of incorrect predictions near the outflow boundaries may also be related to the boundary conditions we set at the outflow. At the outflow, we currently only set the pressure to a value of 0. However, as we discussed previously in section 6.1.1, the boundary condition at the outflow is not derived from nature, and there may be better choices. Specifically, enforcing different boundary conditions, such as $\frac{\partial \vec{u}}{\partial n} = 0$, where

n is the outward-facing vector that approximates the center line of the artery at the outflow boundary, may yield better results. Note that enforcing this particular boundary condition would require approximating the normal vector.

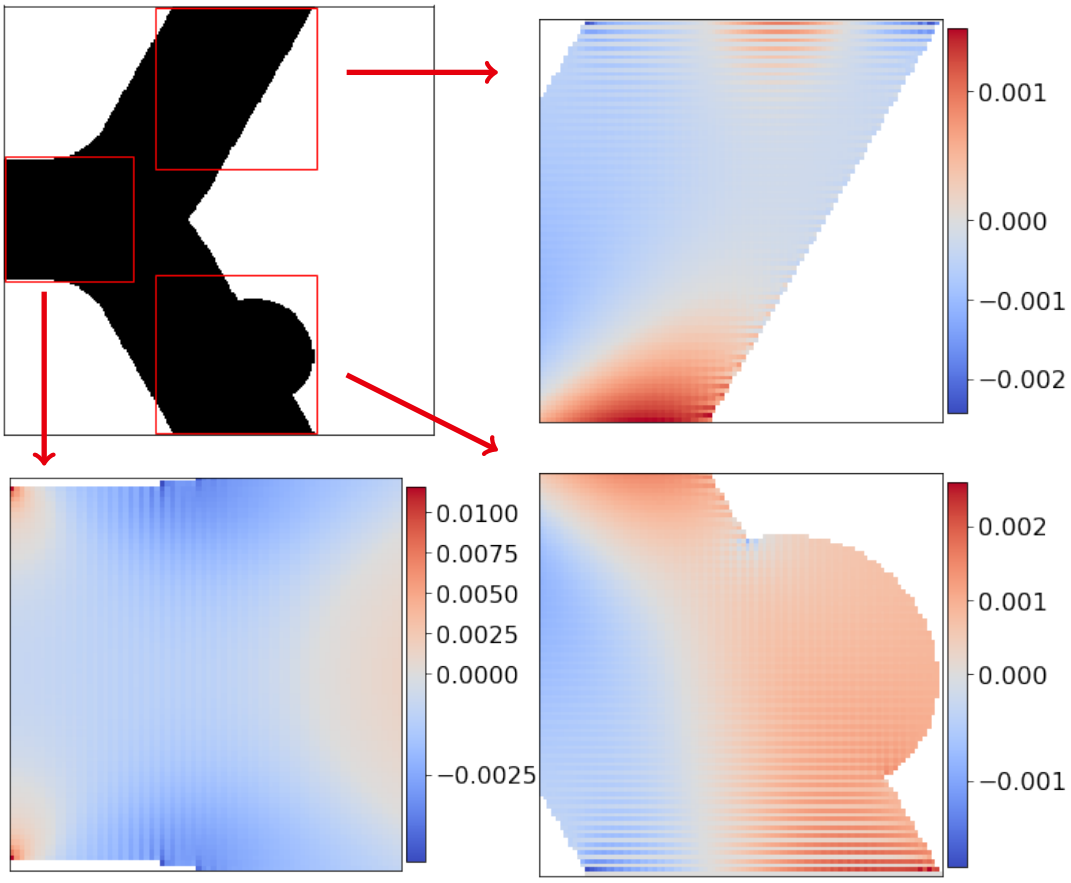


Figure 6.11: A zoom into the pressure near the two outflow boundaries and the inflow boundary.

As a result, we have shown that our method is well-suited for application to non-rectangular geometries. However, we found that, at least for the geometry shown here, oscillations in the pressure become more pronounced. This may be due to the more rasterized nature of the boundary, since if the geometry is not rectangular, a larger portion

of the boundary may not align with the orientation of the uniform grid of the pixel image.

6.2.3 Multiple Channel Geometries

In this section, we apply our method for the first time to a data set consisting of multiple geometries. Here we first investigate the capabilities and limitations of our approach in the actual task of generalizing to geometries not seen during training. We then compare the pure physics-aware approach with the state-of-the-art data-based approach and a mixed approach where we train both the physics-aware and the data-based loss. Per approach, we train multiple models using increasing percentages of geometries from the channel data set as training data and always validate on the same 25% of said data set. The channel data set we consider consists of roughly 5 000 geometries. The obstacles of the geometries in this data set cover a maximum of 50% of the channel in height, keeping a distance of 0.75 from the upper and lower boundary. The maximum velocities for the geometries in the data set are between $3.5\frac{m}{s}$ and $9\frac{m}{s}$.

From our prior work in the previous section, we know that a model that is 8 levels deep is already necessary for a single geometry. Therefore, we do not consider shallower models. Furthermore, we double the number of filters per convolutional layer from 32 to 64 so that our models have sufficient capacity for multiple geometries. For all models considered here, we use ReLU as the activation function and the Adam optimizer with a learning rate of $1 \cdot 10^{-4}$ for training.

6.2.3.1 Physics-Aware Approach

In this section, we first analyze the performance of our models on the entire data set using averaged performance measures. We then examine the predictions on specific geometries. Afterwards, we turn to an analysis of the errors, including the distribution of the errors in our data set.

The performance for the physics-aware approach evaluated on the training and validation

data is listed in table 6.1. Here, the relative L_2 errors of velocity u and pressure p are averaged over the training and validation geometries for four models trained on 10%, 25%, 50%, and 75% of the geometries in the channel data set. For the sake of completeness, we give the mean of the norm of the residuals for the mass equation, denoted $Mass$, and the momentum equations, denoted Mom, also averaged over the geometries. Later we will also examine the distribution of the errors. Let us note that all the models considered were trained for 2 500 epochs. We are now training the model on fewer epochs because we are training it on many more geometries per epoch.

training data		$\frac{\|u_{NN}-u\|_2}{\|u\|_2}$ mean (%)	$\frac{\|p_{NN}-p\|_2}{\|p\|_2}$ mean (%)	$\|Mass\|_2$ mean	$\|Mom\|$ mean	Epochs trained
10%	train	4.34	9.75	$2.8 \cdot 10^{-02}$	$7.4 \cdot 10^{-02}$	2 500
	val	5.70	12.81	$5.7 \cdot 10^{-02}$	$2.0 \cdot 10^{-01}$	
25%	train	4.17	9.61	$2.5 \cdot 10^{-02}$	$6.1 \cdot 10^{-02}$	2 500
	val	4.82	10.73	$4.4 \cdot 10^{-02}$	$1.3 \cdot 10^{-01}$	
50%	train	4.16	9.47	$2.4 \cdot 10^{-02}$	$5.7 \cdot 10^{-02}$	2 500
	val	4.37	9.68	$3.7 \cdot 10^{-02}$	$1.0 \cdot 10^{-01}$	
75%	train	3.82	8.71	$1.8 \cdot 10^{-02}$	$4.0 \cdot 10^{-02}$	2 500
	val	3.91	8.65	$2.8 \cdot 10^{-02}$	$8.0 \cdot 10^{-02}$	

Table 6.1: Performance of the physics-aware approach on multiple geometries from the channel data set compared to OpenFOAM simulations on *locally refined* meshes.

Let us first consider the averaged errors for the velocity u and the pressure p. Here, we can observe that both errors decrease with increasing number of training geometries. For the model trained on the most geometries (75% of the data set are about 3 700 geometries) we are already at a mean error of 3.91% in u and 8.65% in p. In the previous section, we established that we cannot expect relative errors of the order of 1% or less with this

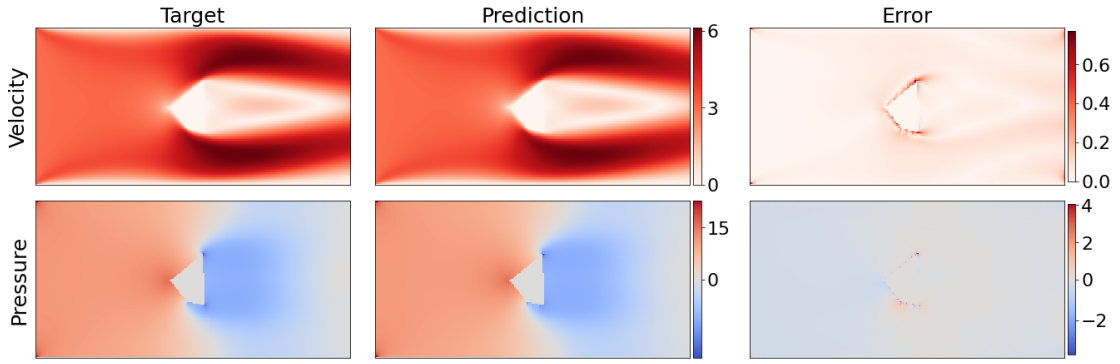
method. For the velocities occurring in this data set, we even expect an error in velocity of over 2%. Considering that the training process here is more challenging than in the previous section, and that we are now aiming to generalize to new, previously unseen geometries, the error values obtained can be confidently described as very good. Of course, what errors are acceptable ultimately depends on the application.

Furthermore, since both training and validation errors are quite similar, we can conclude that there is no overfitting, at least with respect to the averaged results. The discrepancy between the training error and the validation error becomes smaller with increasing number of training geometries and is only 0.09% in u for the 75% model. In the pressure, the model even performs better on the validation geometries than on the training geometries. However, this is most likely due to chance and the distribution of geometries in the training and validation geometries. In the rest of this section, we refer only to the model trained on 75% of the geometries.

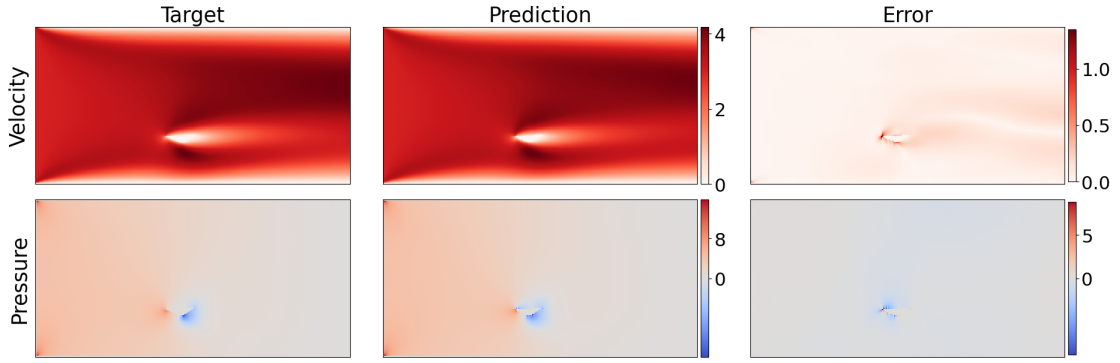
In fig. 6.12, three predictions of our model for different validation geometries are shown. The prediction of our model for the first geometry is one of the best over the whole data set; see fig. 6.12(a). Here the relative error in u is only 1.4% and in p 2.4%. Consistent with this, the error in both pressure and velocity is largest near the obstacle, where we always have a high error due to the misrepresentation of the boundaries.

However, this prediction is not representative of the entire data set. There are geometries where our model performs somewhat worse, but here the predictions are still of high quality and physically reasonable. An example of this is the second geometry; see fig. 6.12(b). This geometry has a very small obstacle and the flow induced by it has a quite low maximum velocity. Again, the prediction of our model is very good, with relative errors of 2.3% in u and 6.0% in p. Here, however, the errors are somewhat higher.

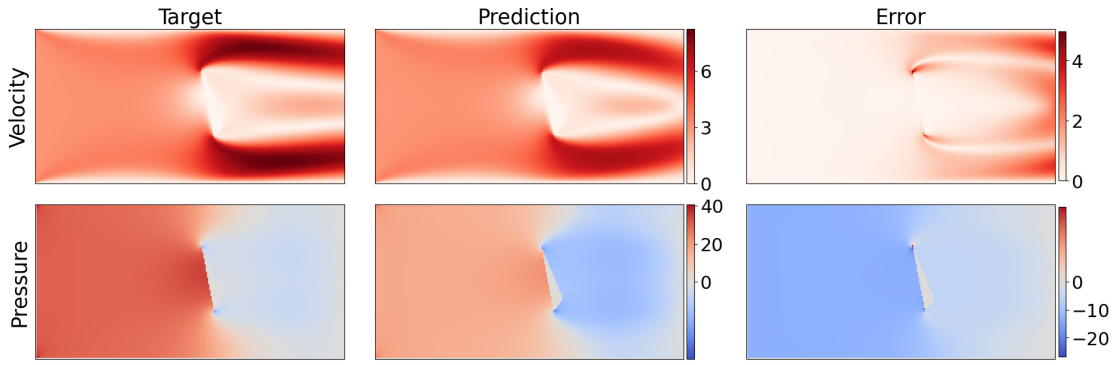
Lastly, there is a class of geometries with large obstacles leading to higher maximum velocities for which our method cannot predict correct flow and pressure fields. The third geometry is such a geometry and the prediction of our model, measured as relative error,



(a) A good prediction. The relative L_2-error in u is 1.4% and 2.4% in p.



(b) Another good prediction. The relative L_2-error in u is 2.3% and 6.0% in p.



(c) The worst prediction. The relative L_2-error in u is 21.0% and 45.8% in p.

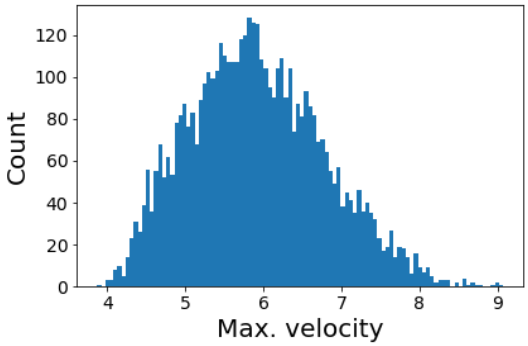
Figure 6.12: Velocity and pressure for the physics-aware approach (Prediction) compared to the OpenFOAM simulation on *locally refined* meshes (Target). The model was trained on 3750 geometries. All shown geometries are validation geometries.

is by far the worst over the entire data set; see fig. 6.12(c). Here the errors are 21.0% in u and 45.8% in p. The prediction obviously does not match the reference solution, as there are massive deviations in the areas around and behind the obstacle. We discuss potential reasons for this later. First we analyze the distribution of the errors in our data set.

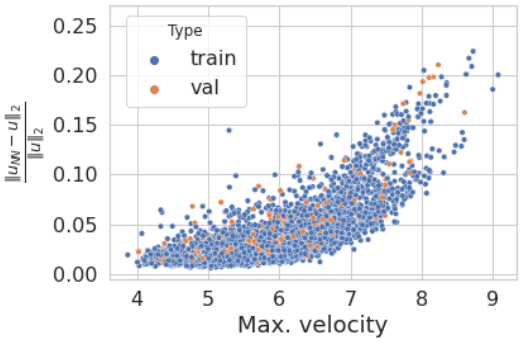
From our preparatory work in section 6.2.1, we know that our method has difficulties with geometries where the limited resolution is not sufficient to allow for a correct flow field to be formed. Therefore, it can be assumed that the error is higher for geometries whose flow field cannot be adequately represented. In fact, when we examine the exact distribution of geometries in terms of the maximum velocity occurring and the relative error in u; see fig. 6.13(b), this assumption is confirmed. Our model seems to be very capable of correctly predicting the flow and pressure when the maximum occurring velocity is below $6\frac{m}{s}$. Here the average errors are 2.5% for u and 5.7% for p. However, as the maximum occurring velocity increases (at least in the reference solutions), so does the error. For geometries with a maximum velocity above $6\frac{m}{s}$, the errors are on average 5.8% for u and 12.7% for p, and there is a clear correlation between the maximum occurring velocity and the relative error in u.

The question immediately arises as to why our approach performs so much worse for higher velocities. There are a variety of reasons for this, and we are not be able to cover all of them. However, we can address some important ones. In particular, we address three possible causes: first the limited resolution, second the choice of the discretization of the convective terms in the governing equations, and third the failure to meet the divergence-free constraint. There are other possible reasons, but we will not discuss them in detail here. For now we note that geometries with larger obstacles are underrepresented in our training data set. However, increasing the number of geometries with larger obstacles did not solve this problem.

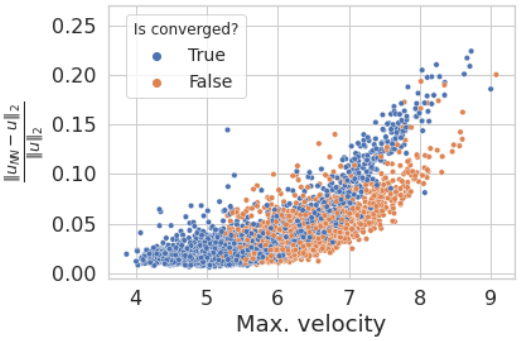
We can, as before with single geometries, try to compare our predictions to simulations on *Cartesian rasterized* meshes instead of *locally refined* meshes. These meshes feature



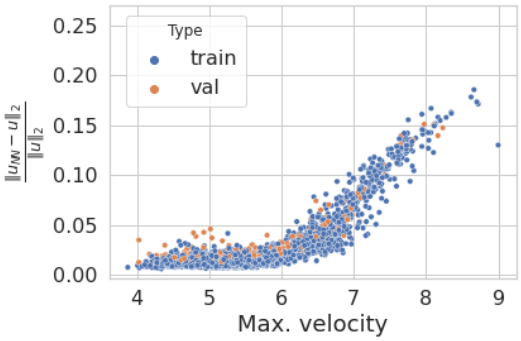
(a) Histogram of maximum velocity in the channel data set.



(b) Distribution of relative error in u w.r.t. simulations on *locally refined* meshes.



(c) Convergence of simulations on *Cartesian rasterized* mesh.



(d) Distribution of relative error in u w.r.t. simulations on *Cartesian rasterized* meshes.

Figure 6.13: Histogram of maximum occurring velocities per geometry in the channel data set (a), relative L_2-error for u for the physics-aware approach compared to OpenFOAM simulations on *locally refined* meshes (b), convergence of OpenFOAM simulations on *Cartesian rasterized* meshes (c), and relative L_2-error for u for the physics-aware approach compared to OpenFOAM simulations on *Cartesian rasterized* meshes (d). The model was trained on 3 750 geometries.

the same erroneous representation of the obstacle and have a similarly low resolution, as is the case with the pixel images used to train our model. By comparing our predictions to simulations on such a mesh we can in some parts account for the discretization error as well as the misrepresentation error.

However, not all simulations on such a low-resolution mesh converge, i.e. reach the relative tolerance set as a stopping criterion. If the flow field becomes too complex and the resolution is no longer sufficient to represent it, classical numerical methods fail to provide correct solutions; cf. section 2.2. Fig. 6.13(c) shows which simulations have converged and which have failed. About a third of the simulations have not converged, most of which are in the range of geometries with maximum occurring velocity greater than $6\frac{m}{s}$. It is not a coincidence that most of the geometries for which the simulation failed to converge are exactly those for which the predictions of our model are increasingly incorrect. This is because we use finite difference stencils to approximate the derivatives and therefore our approach resembles classical numerical methods, i.e., the finite difference method; cf. chapter 5, and as such shares its shortcomings.

We can hower compare the predictions of our model to those simulations on *Cartesian rasterized* meshes that have converged; see fig. 6.13(d). There are three characteristics of this plot that are directly noticeable.

First, for geometries with maximum velocities below $6\frac{m}{s}$ the errors are uniformly low and in particular lower than those compared to the simulation results on *locally refined* meshes. On average, the reduction in error is about 1%. This is the error we can attribute to the incorrect representation of the geometry and the lower resolution. However, it should be noted that the geometries with lower maximum velocities are precisely the geometries with smaller obstacles. And since the obstacles are the only part of the channel geometries that are represented incorrectly, the effect of the incorrect representation of the geometry will be smaller here than for geometries with larger obstacles.

Second, at a maximum occurring velocity above $6\frac{m}{s}$, the errors in the predictions of

our physics-aware model start to increase. This means that this effect is not only due to the low resolution and the erroneous representation of the geometry.

Third, the standard deviation of the errors is significantly lower than before, compare fig. 6.13(b). In addition, there are no more outliers. From both facts it can be concluded that the predictions are more reliable with respect to simulations on *Cartesian rasterized* meshes.

There are two other possible reasons for the poorer performance of our model for higher velocities that we would like to discuss. The first is the choice of discretization of the convective terms in the governing equations. In this section we have discretized all terms in the governing equations with centered second-order differences. However, we know that this is not a suitable choice for the convective terms in the Navier-Stokes equations, especially for the combination of low resolution and higher velocities in the flow; cf. section 2.2.3. Therefore, it may be worthwhile to use other discretizations, such as upwind schemes. We do this in section 6.4.3.

The last possible reason for our model's difficulties at higher velocities that we want to discuss here is the failure to satisfy the divergence-free constraint. Training a physics-aware CNN is essentially a nonlinear optimization task, wherein we seek to minimize the residuals of the nonlinear systems of equations obtained from the governing equations using finite difference approximations. In the basic variant of this approach, we treat the residuals of the discretized divergence-free equation and the discretized momentum continuity equation equally by including both with equal weight in our loss function; see chapter 5. However, this may not be the best approach. Since we are not fully minimizing the residuals, we sometimes allow a solution where there is strong divergence in some regions. Such a solution will of course be wrong. While arguably a solution must satisfy both equations to be considered correct, many classical numerical solution methods for the Navier–Stokes equations include corrector steps to explicitly enforce the divergence-free equation at each iteration step; cf. chapter 2. Therefore, it may be a good idea to consider

a higher weight for the residual of the divergence-free equation in our loss function. We explore this in section 6.4.4.

6.2.3.2 Data-Based Approach

In this section, we briefly analyze the performance of the data-based approach using the channel data set. This analysis is similar to the previous section where we analyzed the physics-based approach, but not as detailed. Afterwards, we compare the two approaches.

training data		$\frac{\|u_{NN}-u\|_2}{\|u\|_2}$ mean (%)	$\frac{\|p_{NN}-p\|_2}{\|p\|_2}$ mean (%)	$\|Mass\|_2$ mean	$\|Mom\|$ mean	Epochs trained
10%	train	2.07	10.98	$1.1 \cdot 10^{-1}$	$1.4 \cdot 10^{-0}$	500
	val	4.48	15.20	$1.6 \cdot 10^{-1}$	$1.7 \cdot 10^{-0}$	
25%	train	1.93	8.45	$9.1 \cdot 10^{-2}$	$1.2 \cdot 10^{-0}$	500
	val	3.49	10.70	$1.2 \cdot 10^{-1}$	$1.4 \cdot 10^{-0}$	
50%	train	1.48	8.75	$9.0 \cdot 10^{-2}$	$1.1 \cdot 10^{-0}$	500
	val	2.70	10.09	$1.1 \cdot 10^{-1}$	$1.2 \cdot 10^{-0}$	
75%	train	1.43	7.30	$1.0 \cdot 10^{-1}$	$1.5 \cdot 10^{-0}$	500
	val	2.52	8.67	$1.2 \cdot 10^{-1}$	$1.5 \cdot 10^{-0}$	

Figure 6.14: Performance of the data-based approach on multiple geometries from the channel data set compared to OpenFOAM simulations on *locally refined* meshes.

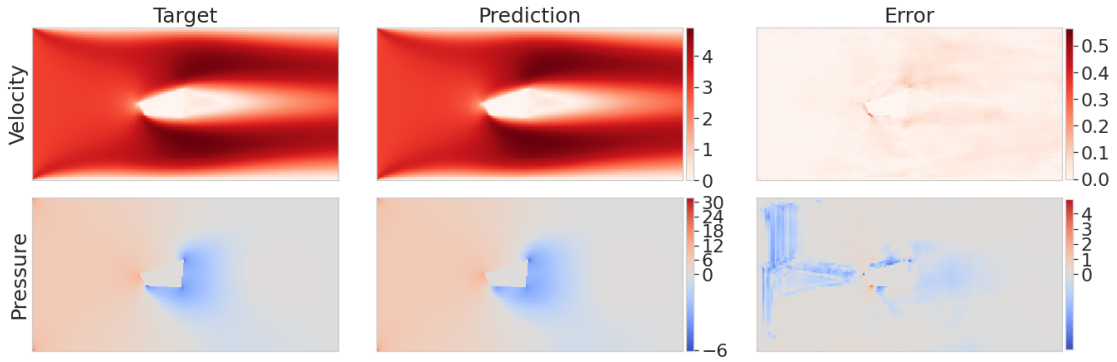
In fig. 6.14, the performance of the data-based approach is evaluated on both training and validation data. The relative L_2-errors for velocity u and pressure p are averaged over four models, each trained on different percentages of geometries from the channel data set. These percentages are 10%, 25%, 50%, and 75%. Additionally, the mean absolute residuals of the Navier-Stokes equations, averaged over geometries, are shown. It should

be noted that all models were trained for 500 epochs, in contrast to the physics-based model, which was trained for $2,500$ epochs. This is because the data-based model is very prone to overfitting and longer training would have reduced the training error but increased the validation error. To avoid this, we stop the training after 500 epochs.

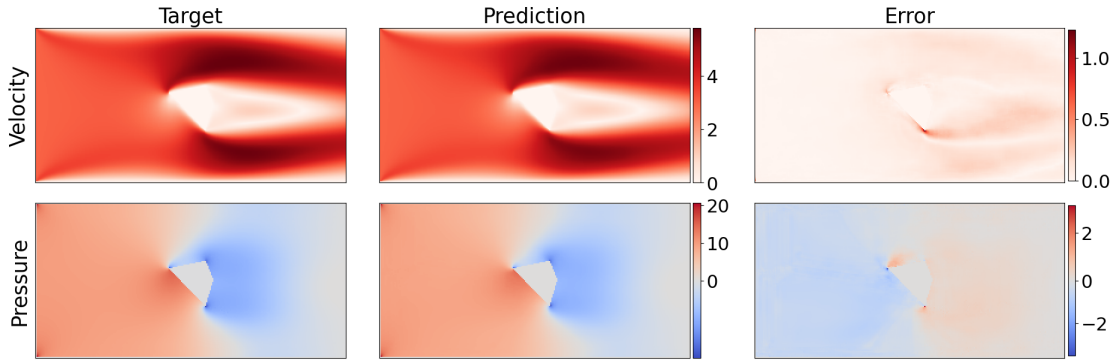
Let us note that the data-based approach also needs less epochs than the physics-aware approach to learn good predictions. This is most likely due to the fact that with the data-based approach, we directly minimize the error between prediction and reference solution, while with the physics-aware approach we minimize the residual of the system of equations; cf. chapter 5.

The data-based models appear to be well capable of learning the flow field and pressure for different geometries. Using just 10% of the geometries as training data, the relative L_2 error on the validation data is as low as 4.48%. With the highest number of training geometries, 75% of the data set, the data-based model achieves relative errors of 2.52% in u and 8.67% in p. However, we see a difference between the errors on the training data and the errors on the validation data for all models. Since the training error is noticeably smaller than the validation error, we can speak of overfitting here. This is extremely undesirable, since we are interested in a model that gives us reasonable and as correct as possible predictions for previously unseen geometries, i.e., a model that can generalize well. Later, we compare the generalizability of this approach with that of our physics-aware approach and again consider this overfitting.

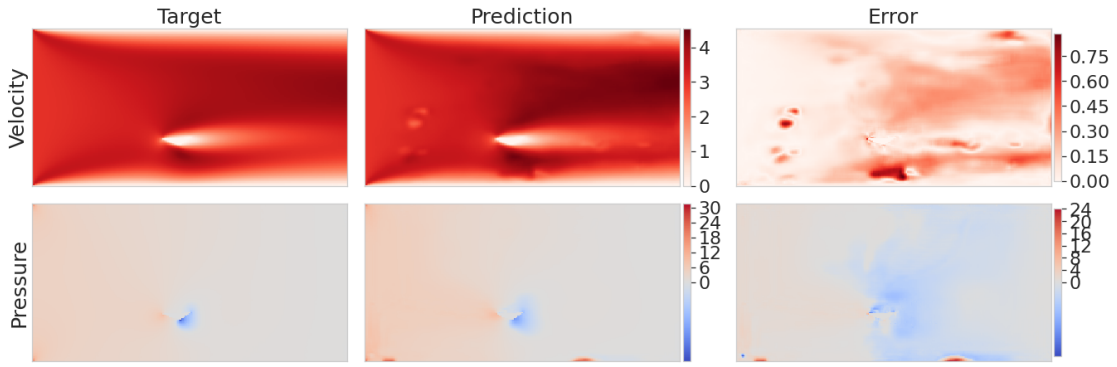
In fig. 6.15 three exemplary predictions of the data-based approach are shown. One of the best predictions is shown in fig. 6.15(a). Here the relative error in u is 1.2% and in p 4.0%. The spatial distribution of the error here is quite random, with the exception of few slightly higher errors in the immediate vicinity of the obstacle, and suggests no systematic error. The second prediction shown in fig. 6.15(b) is exemplary for most of the validation data. The relative error obtained here is also quite low, but slightly higher with 2.4% in u and 6.3% in p. Furthermore, the errors here no longer resemble white



(a) A good prediction. The relative L_2-error in u is 1.2% and 4.0% in p.



(b) Another good prediction. The relative L_2-error in u is 2.4% and 6.3% in p.



(c) A non-physical prediction. The relative L_2-error in u is 5.8% and 34.0% in p.

Figure 6.15: Velocity and pressure for the data-based approach (Prediction) compared to the OpenFOAM simulation on *locally refined* meshes (Target). The model was trained on 3750 geometries. All shown geometries are validation geometries.

noise in their spatial distribution, but occur in the immediate vicinity of the obstacle and at locations where the fluid flows faster. Overall, the data-based approach is very capable of predicting flow and pressure in this setting.

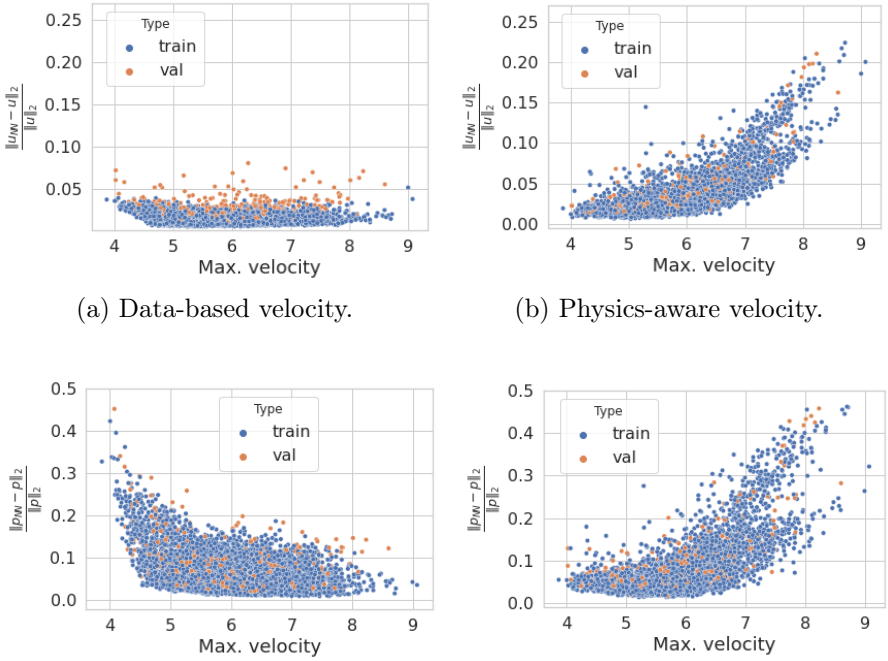
However, some of the predictions made are not physically correct, although the error is not necessarily high. The prediction shown in fig. 6.15(c) is one of them. Here there are nonphysical spots in the flow as well as in the pressure. Nevertheless, the error is not very large with 5.8% in u. In the pressure, however, the error is strongly increased with 34.0%. In this geometry the obstacle is very close to the lower boundary, the distance is only 0.77. In the training data, only obstacles with a minimum distance of 0.75 to the lower and upper boundary were represented. That is, the geometry shown here is at the boundary of the spectrum spanned by the training data. Apparently, the data-based approach has increased problems with geometries lying at the edge of this spectrum.

There is one important aspect to note about the data-based approach. While we train the model on a uniform grid of pixels, the underlying training data is obtained from simulations on irregular meshes adapted to the respective geometry. In particular, these meshes accurately represent the geometry without distorting the exact position and orientation of the obstacle walls, which is not the case with the uniform pixel grid. This means that the training values for velocity and pressure, especially those close to the obstacle, contain more accurate information about the exact position and orientation of the walls than is available from only the pixel image representation of the geometry. Since the data-based approach is presented with this information via the data-based loss, it can potentially extract this information. In contrast, the physics-aware approach cannot access this additional information because it is limited to using only the information available in the pixel image representation of the geometry.

At this point, we refrain from further analysis of the data-based approach on its own and instead move on to compare it with the physics-aware approach.

6.2.3.3 Comparison of Data-Based and Physics-Aware Approach

In this section we compare the data-based approach to the physics-aware approach. Here we consider only the models trained on 75% of the geometries of the channel data set. We analyze and compare the performance of the models first on the level of the entire data set and then on the basis of individual predictions.



(a) Data-based velocity.

(b) Physics-aware velocity.

(c) Data-based pressure.

(d) Physics-aware pressure.

Figure 6.16: Comparison of the relative L_2-error distribution for u and p with regards to the maximum occurring velocity for the data-based ((a) and (c)) and physics-aware ((b) and (d)) approaches compared to OpenFOAM simulations on *locally refined* meshes. Both models were trained on 3 750 geometries.

We have already observed that the average error of the data-based model is significantly lower than that of the physics-based model. However, we also know that the errors of the physics-based model are not equally distributed. Consequently, we examine and compare

the distribution of the errors in the velocity and the pressure for both models. These are plotted in terms of the maximum velocity occurring in fig. 6.16.

The velocity errors of the data-based model, fig. 6.16(a), are consistently low and mostly uniformly distributed, indicating no systematic bias. We see a slight increase in errors on the training data at the *edges* of the training spectrum and increased errors on some validation geometries. Compared to this, the velocity errors of the physics-aware model, see fig. 6.16(b), are almost continuously higher, although they are remarkably lower for low velocities. However, most importantly, they increase linearly above a certain maximum velocity. We have previously discussed that this is likely a problem related to the method of how we approximate the residuals of the governing equations. Let us note that at lower maximum velocities, the performance on the validation geometries of the physics-aware model is equal to that of the data-based model and, in some cases, even better.

A similar pattern can be observed in the pressure errors, see fig. 6.16(d), of the physics-aware model, which also increase somewhat linearly above a certain maximum velocity. This is to be expected since, in the physics-aware model, the velocity and the pressure are coupled through the residual of the momentum equation and an error in the velocity affects the pressure, and vice versa. There is no such coupling in the data-based model. Here the velocity and the pressure are not coupled and only share the encoder part of the CNN architecture; cf. section 4.5. As a result, the distribution of the pressure errors of the data-based model, see fig. 6.16(c), also does not resemble the distribution of the velocity errors. In fact, the pressure errors are comparatively high overall and even higher for low velocities. For low velocities, the pressure difference between the inflow and outflow boundary is also very low in this data set, i.e., the magnitude and range of values of the pressure is comparatively small. Since the velocity and the pressure are not coupled, higher errors in the pressure for lower velocities indicate difficulties of the data-based approach to correctly learn pressure fields of varying magnitudes. Note that we already

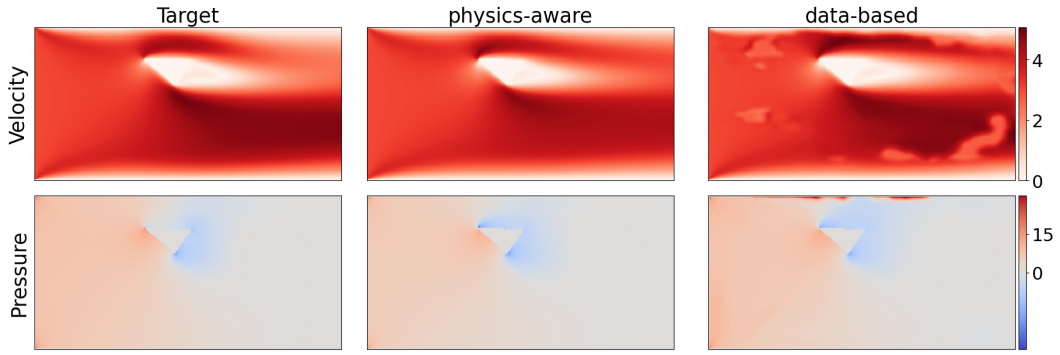
use a scaling technique in the CNN to allow it to work on data of different magnitudes; cf. section 4.5.

Contrary to this, the physics-aware approach seems to have little difficulties with learning the pressure for lower velocities. This is even though a certain part of the error in the pressure is due to the oscillations in the pressure near the obstacle; cf. fig. 6.12. These slight oscillations almost always occur, at least when we employ the basic variant of the physics-aware approach, i.e., in particular, discretizing the pressure terms with centered differences of second order. However, even with these oscillations present, the physics-aware approach outperforms the data-based approach for low velocities, especially with respect to the pressure.

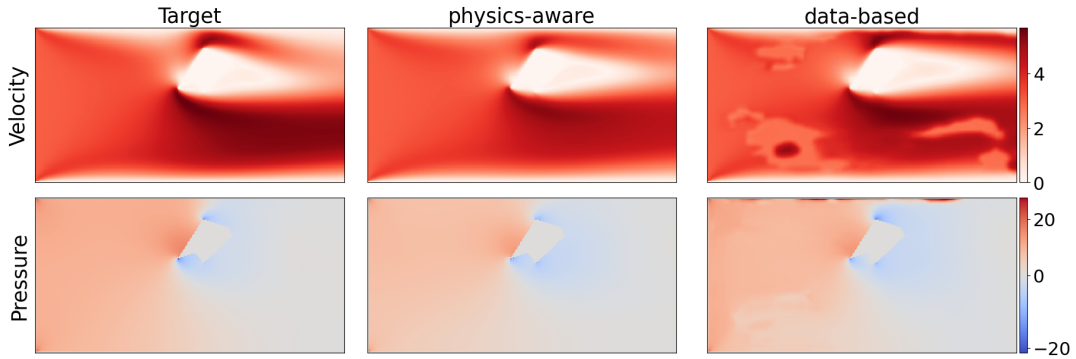
We have seen before that the data-based approach has difficulties with geometries that lie at the boundary of the spectrum spanned by the training data. The training data contains only geometries whose obstacle has a distance of at least 0.75 to the lower and upper boundary. We now consider how good the predictions of the two models are for geometries whose obstacles are closer than 0.75 to the upper boundary. A comparison of the predictions of both models is shown in fig. 6.17. Here the distances of the obstacles to the upper boundary are always smaller than 0.75 and range from 0.63 over 0.4 to 0.2.

We clearly see that the predictions of the data-based model contain strong non-physical errors, both in velocity and pressure. These errors become more intense as the obstacle approaches the upper boundary, i.e., away from the spectrum of the training data. The relative error for u increases from 16% in fig. 6.17(a) then 25% in fig. 6.17(b) to 29% in fig. 6.17(c).

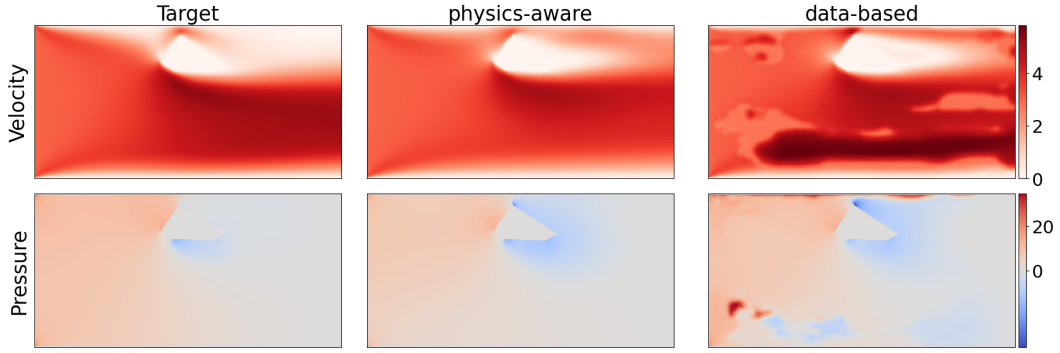
The predictions of the physics-aware model are also not correct, but they are quantitatively and qualitatively better. Thus, no non-physical spots occur here, neither in the velocity nor in the pressure. Also the relative errors are with 10% in, 15%, 21% lower than those of the data-based approach.



(a) The relative L_2-error in u is 10% and 27% in p for the physics-aware and 16% and 37% for the data-based approach.



(b) The relative L_2-error in u is 15% and 31% in p for the physics-aware and 25% and 28% for the data-based approach.



(c) The relative L_2-error in u is 21% and 35% in p for the physics-aware and 29% and 54% for the data-based approach.

Figure 6.17: Comparison of velocity and pressure for the physics-aware and data-based approaches to OpenFOAM simulations on *locally refined* meshes (Target). The distance of the obstacle to the upper wall decreases from 0.63 (a) to 0.4 (b) to 0.2 (c).

6.2.3.4 Combined Approach

Up to now, we have seen that both approaches, the data-based and the physics-aware approach, offer advantages and disadvantages. However, it is possible, if sufficient reference data is available, to train a model using a combination of the physics-aware, see eq. (5.14), and the data-based loss, see eq. (4.6). In this section we explore how such a combined approach performs.

The physics-aware approach, for instance, is not reliant on target data for training and can generalize well to previously unseen geometries. However, the limited resolution of this approach can hinder its ability to learn specific flow fields. Conversely, the data-based approach excels at fitting training data, but faces challenges when attempting to generalize to new geometries. This problem can be partially mitigated by training with massive amounts of data. Hence, the data-based approach may not be suitable when simulation or measurement data are scarce. Additionally, predictions generated by the data-based approach cannot be guaranteed to be physically consistent. Taken together, it appears that the benefits and drawbacks of the two approaches are at least somewhat complementary.

The performance of the combined approach evaluated on the training and validation data is plotted in fig. 6.18. For reference, we also show the performance of the previously described data-based and physics-based approaches. As before, the relative L_2 errors of velocity u and pressure p are averaged over the training and validation geometries for four models trained on 10%, 25%, 50%, and 75% of the geometries in the channel data set. Let us note that all the combined approach models were trained for 2 500 epochs.

Based on the errors averaged over the training and validation data, we can state that the combined approach consistently performs better than the data-based and physics-aware approaches. While the improvement in the velocity is minimal, it is surprisingly large in the pressure. For the model trained on 75% of the geometries, the error with the combined approach on the validation data is only 4.08%. In comparison, the data-based

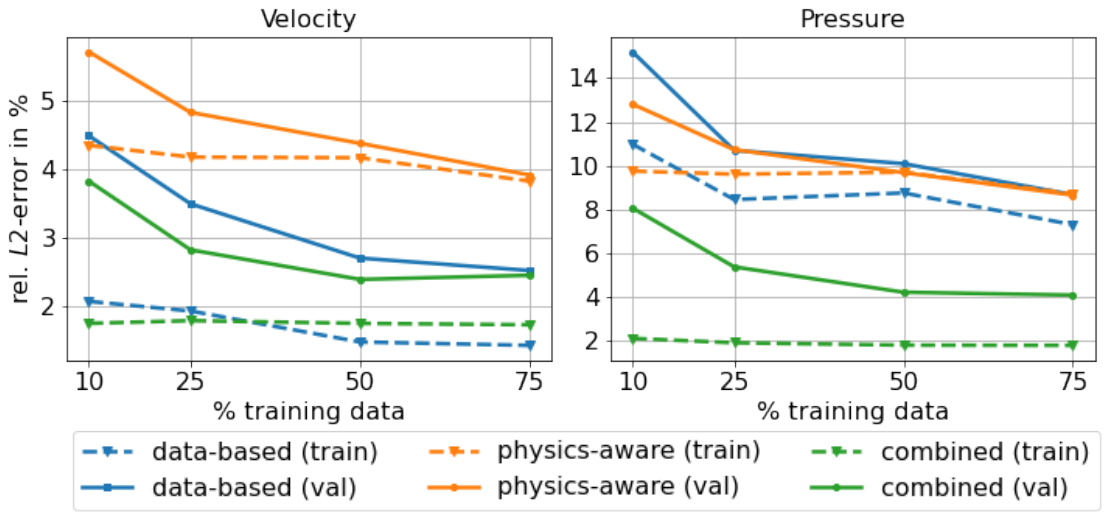
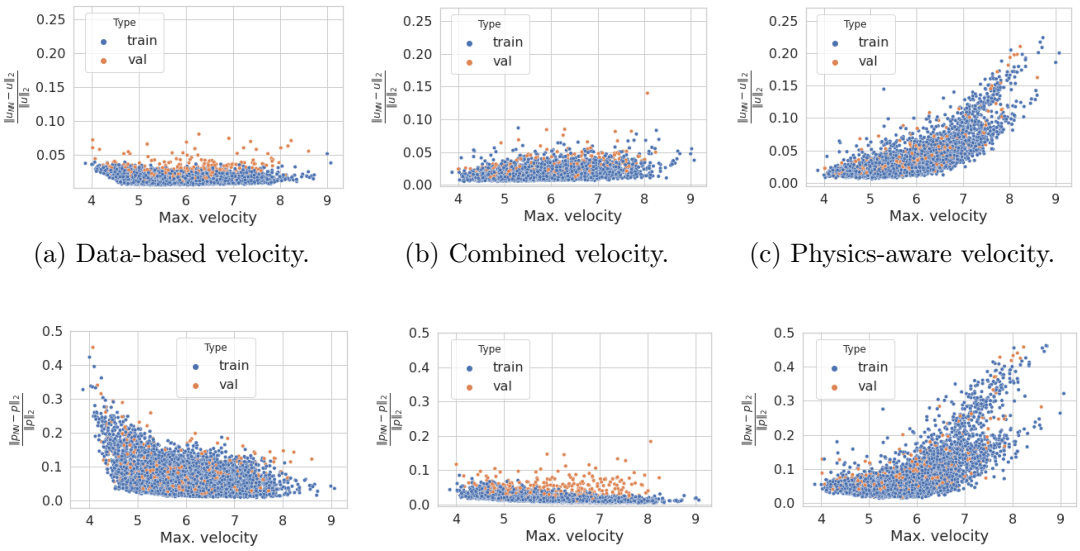


Figure 6.18: Performance of the data-based, combined, and physics-aware approaches on multiple geometries from the channel data set compared to OpenFOAM simulations on *locally refined* meshes.

approach here is 8.67% and the physics-aware approach is 8.65%.

Statements based on values that have been averaged over a large number of geometries are handy and quick to assess, but they are only meaningful to a limited extent. For this reason, we again consider the distribution of errors in velocity and pressure with respect to the maximum occurring velocity, depicted in fig. 6.19.

The distribution of the velocity errors of the combined approach, see fig. 6.19(b), is similar to that of the data-based approach, see fig. 6.19(a), with the difference that the errors on the training data are slightly higher. However, since the errors on the validation data are not increased and the distributions of the errors on the training data and validation data are more similar, the higher errors on the training data are not a disadvantage. Similar errors on the training and validation data are desirable, as this indicates an absence of overfitting. In particular, the error distribution of the combined approach for low maximum velocities is very similar to the error distribution of the



(a) Data-based velocity. (b) Combined velocity. (c) Physics-aware velocity.

(d) Data-based pressure. (e) Combined pressure. (f) Physics-aware pressure.

Figure 6.19: Comparison of the relative L_2-error distribution for u and p with regards to the maximum occurring velocity for the data-based ((a) and (d)), combined ((a) and (e)) and physics-aware ((c) and (f)) approaches compared to OpenFOAM simulations on *locally refined* meshes. All models were trained on 3 750 geometries.

physical approach, see fig. 6.19(b). However, at higher maximum velocities, the errors of the combined approach remain consistently low.

Of particular interest are the errors in pressure. We have previously noted that the data-based approach, due to the lack of coupling between velocity and pressure in the loss terms, has difficulty with learning correct pressure fields, as can be seen by the high errors in fig. 6.19(d). The physics-aware approach, on the other hand, in the version applied here with centered differences of second order, equal weighting of the loss terms; cf. chapter 5, has difficulties in general with higher velocities, not only in terms of the pressure, see fig. 6.19(b) and fig. 6.19(e). In contrast, the errors in the pressure of the combined approach are very low over all maximum occurring velocities; see fig. 6.19(f).

In fact, they are lower throughout than for the other two approaches.

6.2.4 Multiple Artery Geometries

As in the case of single geometries, we now apply our method to non-rectangular geometries. For this purpose, we consider two data sets whose geometries each have different types of aneurysms. The first data set consists of bifurcation geometries and the second of single artery geometries; cf. section 2.3.

6.2.4.1 Bifurcation

We train a model on 500, 1 500, and 3 500 geometries, respectively. The associated averaged errors over the training and validation data are shown in fig. 6.20.

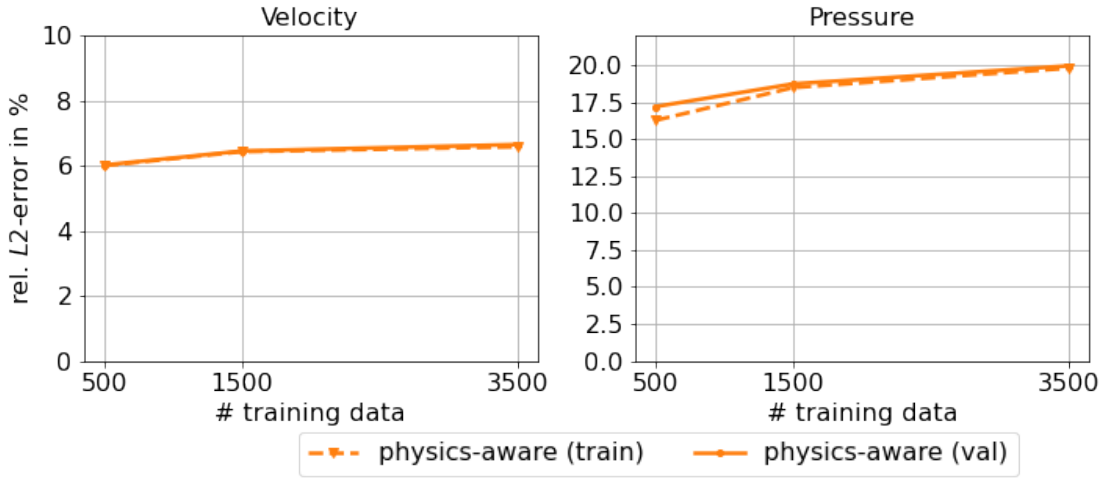
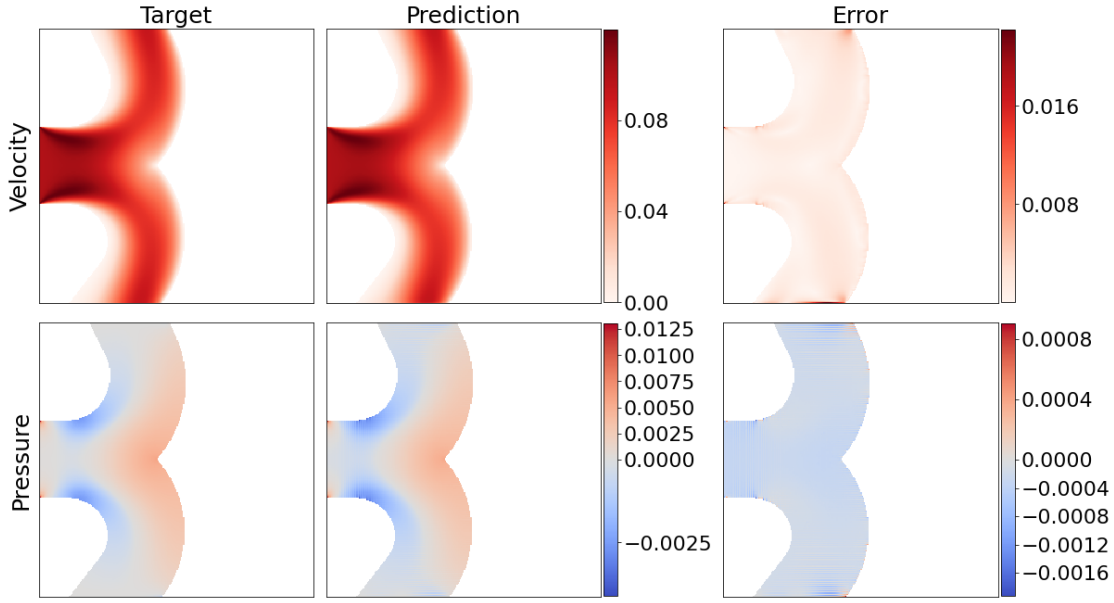
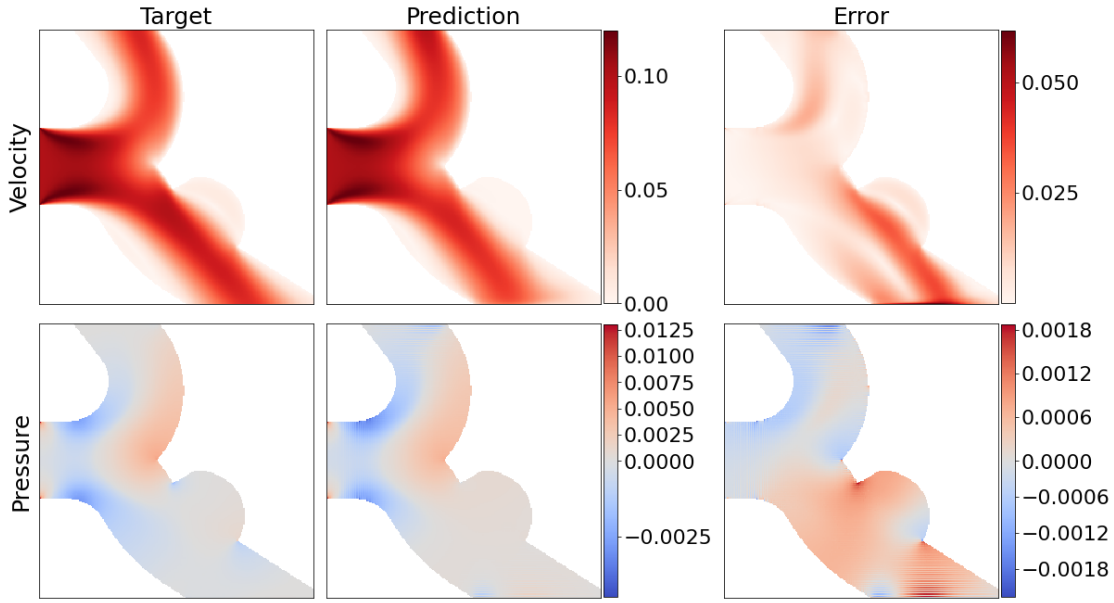


Figure 6.20: Performance of the physics-aware approach on multiple geometries from the bifurcation data set compared to OpenFOAM simulations on *locally refined* meshes.

It is immediately noticeable that the errors do not decrease for larger training data. The error in pressure even increases with more training geometries. Such behavior is



(a) A good prediction. The relative L_2-error in u is 2.3% and 13.0% in p.



(b) A bad prediction. The relative L_2-error in u is 19.8% and 35.6% in p.

Figure 6.21: Velocity and pressure for the physics-aware approach (Prediction) compared to the OpenFOAM simulation on *locally refined* meshes (Target). The model was trained on 3 500 geometries. All shown geometries are validation geometries.

not desirable and is indicative of a fundamental problem. Nevertheless, compared to the result on a single geometry from section 6.2.2, the obtained average errors are rather good with 6.4% in u and 19.5% in p. However, the errors are not equally distributed. Therefore, we now consider individual predictions.

In fig. 6.21, two predictions of our model are plotted. The first one in fig. 6.21(a) is one of the best predictions on the validation data set. Here the errors in u and p are only 2.3% and 13.0%, respectively. The second prediction in fig. 6.21(b) is the worst prediction on the validation data set. Here the errors in u and p are 19.8% in u and 35.6% in p.

At least three conclusions can be made based on these predictions. First, our physics-aware approach is in principle able to handle non-orthogonal geometries even in the case of multiple geometries. In particular, the location and size of the outflow boundaries vary in this data set. The predictions generally fit the reference data well, and the model responds correctly to variations in geometry. Second, strong pressure oscillations occur near the inflow and outflow boundaries, as already in the case of the simple geometry. Especially near the outflow boundaries, these errors in the pressure lead to distortions in the flow field. This effect is particularly evident in the second prediction, fig. 6.21(b). Here, the oscillations in the pressure near the lower outflow lead to an erroneous pressure and, consequently, to an erroneous flow field. Third, the predictions of our model are flawed near the aneurysm. This could be due to the pressure oscillations, since the aneurysm is always relatively close to the outflow boundary. However, it could also be a general weakness of our approach; or it could be due to the composition of our data set, as it does not contain many aneurysms that have a large influence on the flow.

However, we can check more than just the predicted velocity and pressure. Since our model is trained by minimizing the residuals of the governing equations, these residuals are also computed automatically. Looking at the residuals of the two predictions, see fig. 6.22, we find that the residuals near the aneurysm are higher for the second prediction. This indicates that the solution in this area is problematic. A high $Mass$ residual in a

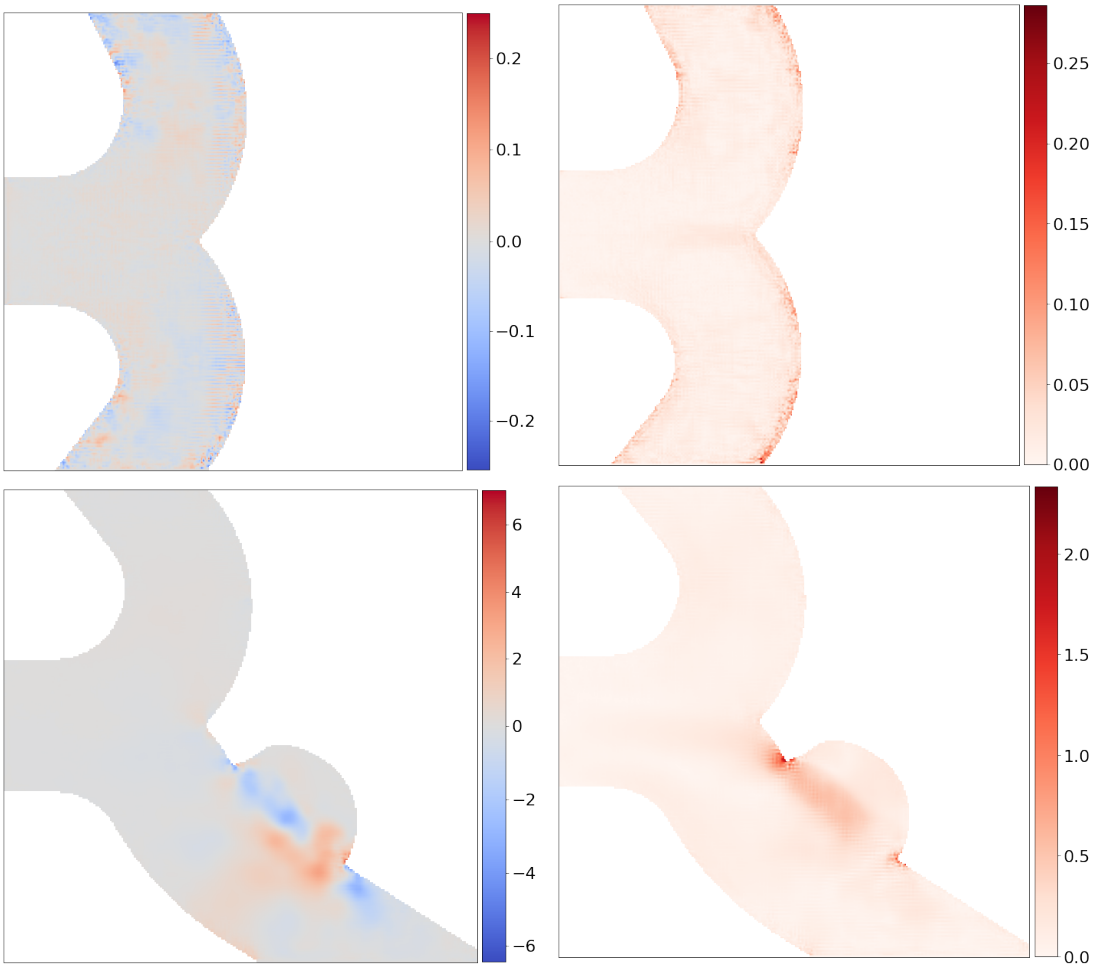


Figure 6.22: Pixelwise residuals $Mass$ (left) and $\|Mom\|_2$ (right) for the predictions shown in fig. 6.21.

grid node or pixel means that more fluid is flowing in than out at that point, or vice versa for a negative $Mass$ residual. Consequently, it is to be expected that the predicted solution is faulty if the $Mass$ residual is high. For the first prediction, the residuals are lower overall, and areas with comparatively higher residuals correspond to areas where the error in velocity and/or pressure is also higher, especially near the arterial walls.

6.2.4.2 Single Artery

Our investigations with the previous data set have shown that our approach works in principle also in the case of multiple geometries on non-rectangular geometries. However, we found that using second-order finite differences to compute the residuals on these geometries can lead to strong oscillations. In section 6.4, we will explore some modifications to our approach, some of which are specifically aimed at avoiding these oscillations. At this point, we foreshadow that section somewhat by using tenth-order finite differences to compute the residuals. By doing so, we locally introduce a stronger coupling of the solution variables and reduce the possibility of oscillations. At the same time, the truncation error of the higher-order finite differences is smaller.

In this section, we consider the behavior of our models for single artery geometries, a geometric configuration where the existing geometric variation is more focused on the aneurysm than in the previous case; cf. section 2.3. We only present results for a model that we trained on 3 500 of these geometries. There are no reference solutions for the training data set, so we cannot report errors on the training data in this case. We evaluate this model on a validation data set of about 300 geometries.

The errors averaged over the validation data set are 9.6% in u and 11.8% in p. However, the errors are far from evenly distributed. In velocity, the errors range from 2% to 50%, with a standard deviation of 8%; and in pressure from 2% to 40%, with a standard deviation of 7%. Low errors are significantly more common than high errors.

In this data set, unlike in the channel data set, see section 6.2.3, there is not much difference in the maximum velocity. This means that the errors are most likely not related to numerical problems, but to variations in the geometries. To find out exactly what caused the higher errors, we compare the predicted velocity for a set of geometries in which we vary only one of the variable parameters. Thus, in fig. 6.23 we show predictions for geometries with a small aneurysm placed approximately in the center of the bend and a bend angle between 0° and 130°; in fig. 6.24 for geometries with a bend angle of 90°,

aneurysm placement in the anterior third of the arc, and varying aneurysm size; and in fig. 6.25 for geometries with a bend angle of 90° and a large aneurysm whose placement varies along the arc.

The predictions for the variable bend angle geometry, see fig. 6.23, show that the model has no difficulty with the variation of the bend angle. The velocity error of these predictions are 2.8%, 6.4%, 2.4% and 8.6%. In all predictions, there is an increase in the errors near the outflow, as observed previously for other geometries. However, these errors are not very large and are at most within a tenth of the magnitude of the outflow. In addition to the relative error calculated over the entire geometry, we can also compare the errors calculated only over a specific part. In this case, we compare the errors calculated in a bounding box of the aneurysm. The bounding boxes are shown in black in the comparison plots. Here, the velocity errors in the aneurysm are 18.5%, 8.8%, 10.9%, and 26.2%. Except for the first error, these errors increase. The first error is larger because very little fluid enters the aneurysm at a bend angle of 0°, and at low velocities in the aneurysm the relative error quickly becomes large. As the bend angle increases, the influence of the aneurysm on the fluid flow becomes greater and the prediction errors near the aneurysm become larger. This is particularly evident in the final prediction for the 130° geometry.

This relation is even more evident when we compare the predictions for variable aneurysm size; see fig. 6.24. Here, the aneurysm size is varied from 0% to 50%, 75%, 100%, and 130%. The relative error in velocity varies from 1.9% to 2.4%, 5.3%, and 11.5% to 22.4%. The relative velocity error in the bounding box of the aneurysm even goes from 13.0% to 19.9% and 28.8% to 43.2%. Note that there is no error inside the aneurysm for the first geometry because there is no aneurysm. From the relative errors, it is already clear that the trained model has difficulty determining the influence of larger aneurysms on fluid flow. Moreover, the mean absolute values of the mass residual and the momentum residual increase with increasing aneurysm size from $8 \cdot 10^{-2}$ to $1.2 \cdot 10^{-1}$ and

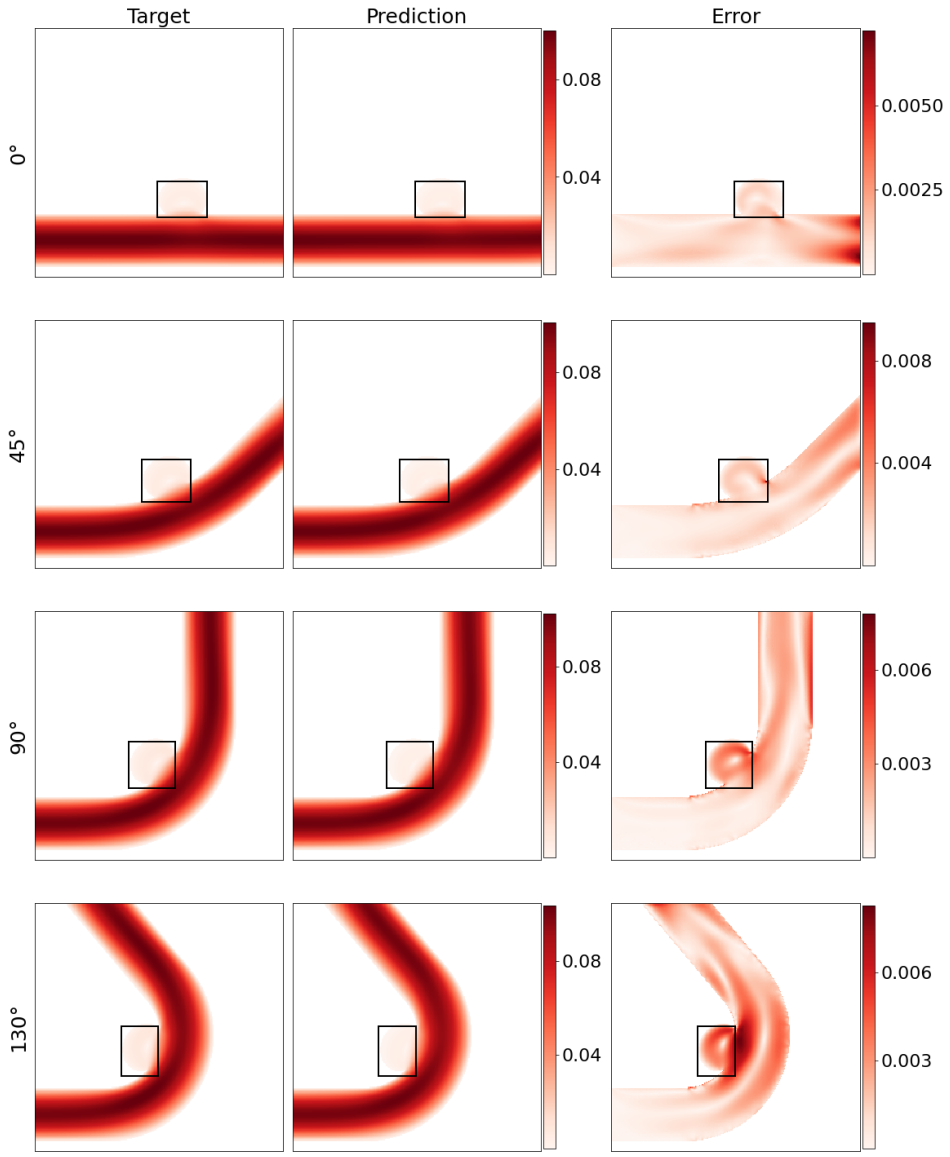


Figure 6.23: Comparison of the predicted velocity for geometries with varying outflow degrees.

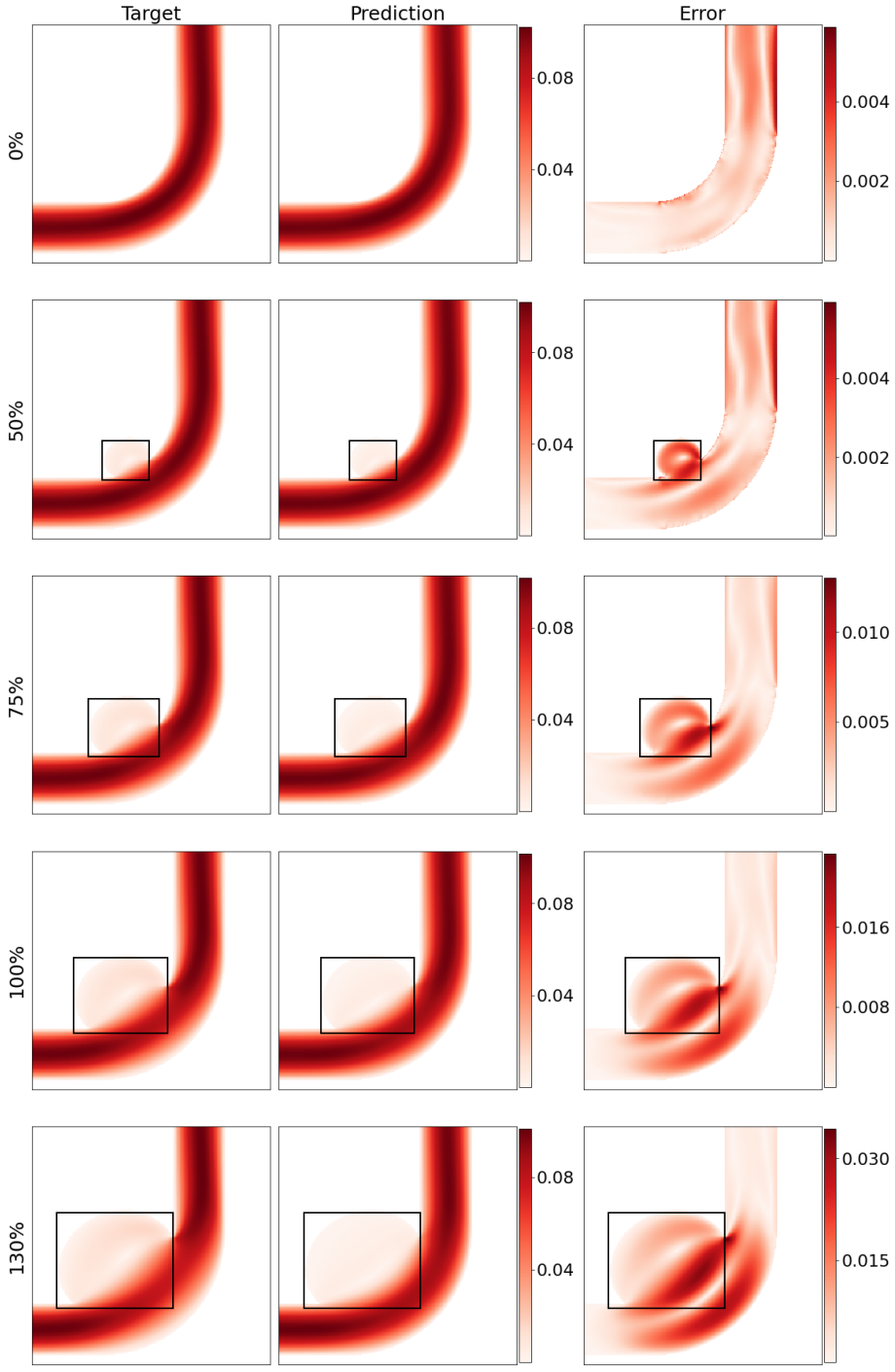


Figure 6.24: Comparison of the predicted velocity for geometries with varying aneurysm sizes.

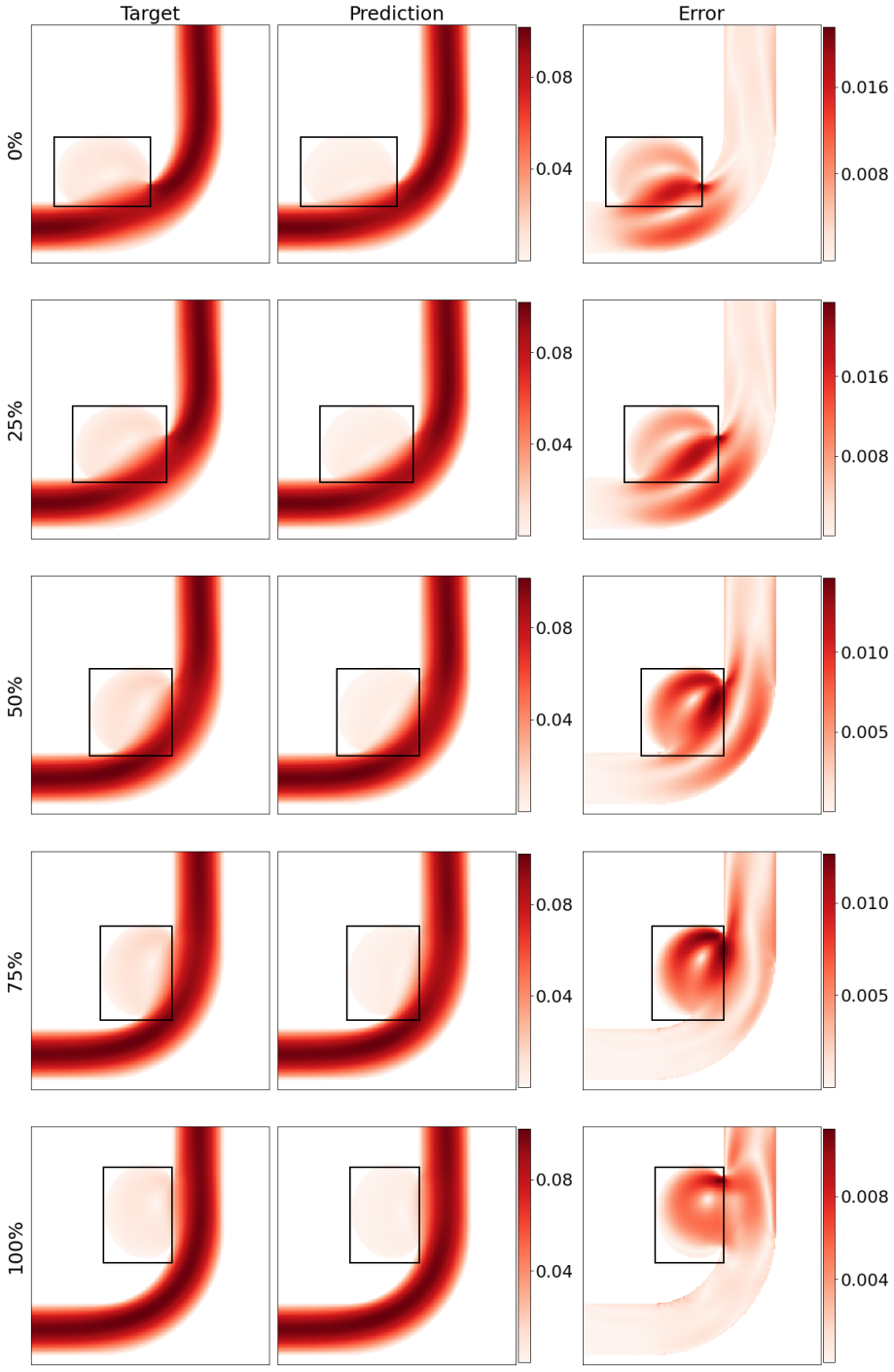


Figure 6.25: Comparison of the predicted velocity for geometries with varying aneurysm placements.

from $3 \cdot 10^{-2}$ to $1.2 \cdot 10^{-1}$, respectively. At the same time, however, the mean absolute values of mass residual and momentum residual decrease slightly in the bounding box of the aneurysm. Here they decrease from $2.4 \cdot 10^{-1}$ to $1 \cdot 10^{-1}$ and from $1.6 \cdot 10^{-1}$ to $1.1 \cdot 10^{-1}$, respectively. Note, however, that this may be due to the fact that as the size of the aneurysm increases, more and more of the artery is included in the rectangular bounding box.

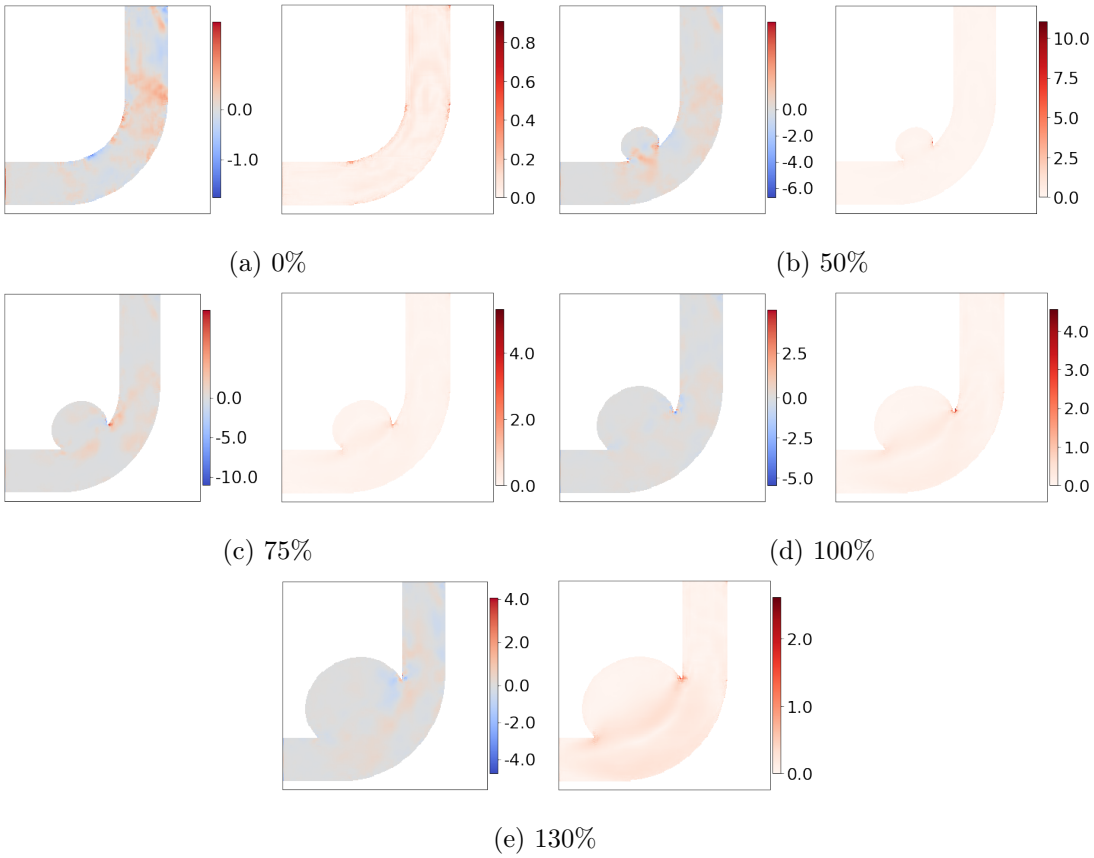


Figure 6.26: Pixelwise residuals $Mass$ and $\|Mom\|_2$ for the predictions shown in fig. 6.24. The geometries have aneurysms with varying sizes

Therefore, we show in fig. 6.26 the discussed residuals that belong to the predictions shown in fig. 6.24. It is clear from these plots that as the size of the aneurysm increases, a

problematic area becomes increasingly visible. At the posterior corner where the aneurysm rejoins the artery, both the mass and momentum residuals are greatly increased. In addition, it can be seen that the predictions for the two largest aneurysms also have higher momentum residuals along the entire transition from the aneurysm to the artery. Therefore, it may be a good idea to increase the weight of the residuals in these areas when calculating the physical loss. However, one must be careful not to give more weight only to the residuals in the aneurysm, otherwise the error may simply be shifted toward the artery. This is because locally in the aneurysm a velocity of 0 corresponds to a minimum in the residuals.

Finally, we see from the predictions with variable aneurysm placement, see fig. 6.25, that the quality of the predictions is independent of the placement of the aneurysm. Here, the velocity error is 9.2%, 11.5%, 8.1%, 6.6%, and 4.6%, respectively. Thus, while there is some effect of aneurysm placement on the error, with the error being highest when the aneurysm is placed just after the start of the curve, the size of the aneurysm has a much greater effect.

6.3 Varying Boundary Conditions

Up until now, we have maintained constant boundary conditions for simplicity's sake. This means that, for example, the inflow velocity remained constant at $3\frac{m}{s}$. However, the pixel-image based CNN approach offers a significant advantage in that it is not bound by this limitation. In this section, we examine how our physics-based approach performs when trained for varying boundary conditions, focusing on the example of inflow velocity. This involves introducing a new variation, the inflow velocity, in addition to the existing variation of geometries, which raises the problem's complexity.

It is worth noting that PINNs and similar approaches face difficulties with this task. For example, Cuomo et al. mention in [30] that PINNs suffer from the disadvantage of requiring a new model to be trained every time the boundary conditions change. We

demonstrate in the following that our approach does not suffer from this drawback. It should be noted that we could, of course, apply the procedure presented below to other variations, e.g., in the material parameters.

First, the architecture of the CNN has to be extended by an additional input to be able to process the variations in the boundary conditions. Since the input to the CNN already consists of a pixel image of the geometry, it makes sense to pass the boundary conditions in the form of a pixel image as well. In this section, we only consider the variation of the inflow velocity. For simplicity, we continue to assume a constant inflow velocity, which we vary between $0.5\frac{m}{s}$ and $5\frac{m}{s}$. The simplest way to convert this variation into a pixel image is to create a second pixel image of the geometry in which the pixels contain the constant inflow velocity as a value.

This is just a design choice, and others are possible as well. At this point, we are only interested in demonstrating the ability of our approach to handle variations in the boundary conditions, not in determining the best way to do so. In the same manner, we could handle variations in other boundary conditions or even material parameters. However, this would further increase the complexity of the problem, so we restrict ourselves to variation of the inflow velocity.

In practice we would treat the geometry pixel image and the inflow velocity pixel image as two channels in the input layer and would thus only have to change the input layer. Note that extending the input in this manner doubles the parameters of the first layer, since the convolutions now affect two channels instead of one. This marginally increases the size of the CNN if all other hyperparameters are kept unchanged.

As mentioned before, in this section we consider a variation of the inflow velocity between $0.5\frac{m}{s}$ and $5\frac{m}{s}$. For the validation data, we create a new data set with geometries similar to those from the channel data set, but with inflow velocities uniformly drawn from the range $[0.5, 5.0]$. Therefore, the error values we get in this section are only partially comparable to those in the previous section. We continue to use the geometries from

our channel data set as training data. For each geometry, we uniformly choose an inflow velocity from an interval $[v_{\min}, v_{\max}] \subset \mathbb{R}$ before training.

Note that this results in each geometry being associated with a single inflow velocity during training. We could further improve the diversity of geometry and inflow velocity pairs by randomly sampling the inflow velocity during training. This would result in each geometry being associated with a different inflow velocity at each epoch during training. However, we do not consider this advanced sampling strategy in this thesis.

We consider the previous standard case where we keep the inflow velocity constant at $3\frac{m}{s}$, and then models where we extend the interval from which we sample the training inflow velocities to $[2, 3]$, $[1, 3]$, and finally $[1, 4]$. We train two models, one on 1 000 and another on 4 500 geometries, for all four ranges.

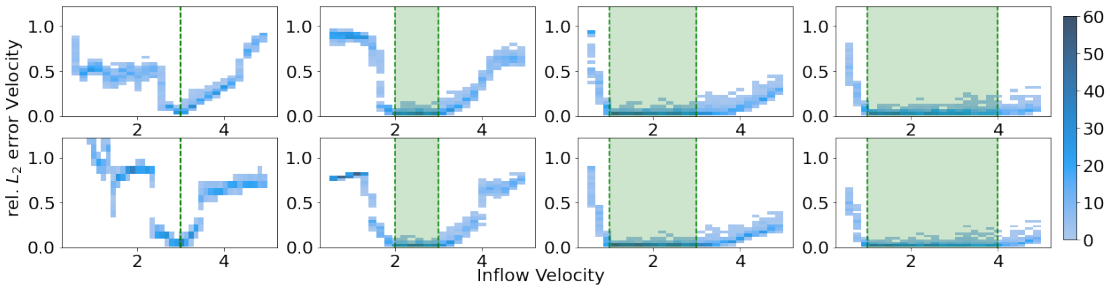


Figure 6.27: Histograms of the relative L_2-errors of the velocity for physics-based models trained on varying ranges of inflow velocities (green area). We show results for 1 000 training geometries (top row) and for 4 500 training geometries (bottom row).

The corresponding errors in the velocity are plotted as a histogram in in fig. 6.27 and the mean values over bins of inflow velocities are listed in table 6.2. Clearly, the model trained exclusively with inflow velocities of $3\frac{m}{s}$ is not able to make valid predictions for other inflow velocities. However, as the range of inflow velocities used during training is expanded, the ability of the models to make valid predictions for other inflow velocities

# Data	Range I.v.	[0.5, 1.0]	[1.0, 2.0]	[2.0, 3.0]	[3.0, 4.0]	[4.0, 5.0]
1 000	[3.0, 3.0]	55.5%	48.1%	31.1%	17.4%	61.5%
	[2.0, 3.0]	89.3%	57.4%	4.0%	15.5%	59.1%
	[1.0, 3.0]	40.2%	3.8%	4.3%	7.1%	20.4%
	[1.0, 4.0]	31.3%	4.0%	4.3%	5.8%	7.7%
4 500	[3.0, 3.0]	186.8%	87.1%	40.5%	36.9%	70.6%
	[2.0, 3.0]	78.4%	44.3%	3.2%	16.1%	68.2%
	[1.0, 3.0]	38.7%	2.9%	3.4%	6.7%	18.5%
	[1.0, 4.0]	27.7%	3.1%	3.4%	4.7%	7.2%

Table 6.2: Mean relative L_2-errors of the velocity for physics-based models trained on varying ranges of inflow velocities

improves. For example, the models trained with inflow velocities in the range $[2.0, 3.0]$ are very capable of making valid predictions for the validation geometries whose inflow velocity is also in the range $[2.0, 3.0]$. Here, the average velocity errors are comparable to the errors obtained in the previous sections.

The predictions of these models for validation geometries whose inflow velocity is not in their training range but in a different range, for example in $[1.0, 3.0]$ and $[3.0, 4.0]$, are not satisfactory. However, it is noticeable that the errors for geometries with a higher inflow velocity are typically lower than for geometries with a lower inflow velocity. This is most likely due to the fact that the geometries with a small obstacle and a higher inflow velocity yield a similar flow field to a geometry with a larger obstacle and a lower inflow velocity. In contrast, the flow field in a geometry with a small obstacle and a lower inflow velocity does not resemble any flow fields in a geometry with any obstacle and a higher inflow velocity.

Both trends continue as we broaden the range from which we sample the training inflow

velocities. As we increase the range the errors on the validation geometries with inflow velocities within this range decrease.

At this point, it should be noted that if we increase the range from which we take the training inflow velocities without increasing the number of training data, fewer values will be taken from a given range. For example, the model trained on 4 500 geometries with inflow velocities from the range $[2.0, 3.0]$ saw exactly 4 500 inflow velocities from that range during training. In contrast, the model trained on 4 500 geometries with inflow velocities from the range $[1.0, 3.0]$ saw only about half of that, approximately 2 250 inflow velocities from the range $[2.0, 3.0]$. Nevertheless, the error obtained on validation geometries with inflow velocities from the range $[2.0, 3.0]$ does not increase or increases marginally by 0.2%, as we extend the range from which we sample. This is a very interesting observation, as it implies that we can extend the applicability of our model to varying inflow velocities at virtually no, or at least little, cost, at least for the use case considered here.

In this section, we have shown that we can easily extend our model to handle variations in the boundary conditions. Extending the range from which we sample the training inflow velocities does not reduce the ability of the model to make predictions for the previous, smaller range of inflow velocities. We do not claim that the results presented here are optimal, and it is most likely possible to improve on them.

6.4 Modifications

We have demonstrated the capabilities and limitations of our approach using a *basic version* in section 6.2. Here, we use the term *basic version* to refer to the procedure of approximating the partial derivatives with centered differences of second order, calculating the residuals with the standard (unmodified) Navier–Stokes equations and weighting the loss terms of the two equations equally. This procedure is described in chapter 5.
In this section we address some of the previously observed limitations. In section 2.2.3, we discussed some problems that can arise when solving the incompressible Navier–

Stokes equations numerically using the finite difference method, and we also discussed possible solutions to these problem. Because our approach essentially entails discretizing the Navier–Stokes equations using finite differences and solving the resulting system of equations with an improved variant of the gradient descent algorithm, we can assume that these problems also occur here. Moreover, it is a reasonable assumption that the solutions discussed there will also work with our approach. However, since our method prominently features a CNN as a surrogate model for the grid functions of the solution variables, it is unclear what the implications of this are and what effect the proposed modifications will have.

In fact, we have observed some of these problems in the previous sections. For example, we often observed oscillations in the pressure. We mentioned that this could possible be due to our usage of centered differences as well as the failure to satisfy the inf-sup condition; cf. section 2.2.3. To address the first cause, we utilize finite differences of higher order in section 6.4.1. Although these differences are still centered, the increased stencil size causes the pressure to be coupled across multiple nodes, thus dampening oscillations. Additionally, with higher order differences we reduce the truncation error, and it is interesting to see if this improves predictions in general. As for the second reason, we cannot easily satisfy the inf-sup condition on an unstaggered grid. However, we can try to get around satisfying the condition by inserting a small stabilization term in the divergence-free equation; cf. section 2.2.3. We present results for this in section 6.4.2.

Furthermore, in section 6.2 we found that our approach has problems with predictions for geometries that induce faster flow velocities, while it has few problems with other geometries that lead to slower flow velocities. Since the cell Reynolds-number Re_h is higher and probably above 2 for faster flow velocity and constant resolution, we could possibly solve this problem by employing upwind schemes. To this end, we employ some upwind schemes in section 6.4.3.

Further, we investigate whether there is an effect of increasing the weight of the loss

term corresponding to the mass equation; see eq. (2.2). Sometimes the mass equation is referred to as divergence-free equation, condition, or constraint; and in most iterative solution methods for the incompressible Navier–Stokes equations, the divergence-free constraint is enforced in each iteration step. In our approach, however, the divergence-free constraint is not necessarily satisfied. At the beginning of the training, it can even be strongly violated due to the randomized initialization of the CNN. Therefore, it is of interest to examine the effect of a higher weighting of the mass residual. We investigate this in section 6.4.4.

In principle, it is not clear for any of the considered modifications that their implementation will lead to improvements in the predictions. Some modifications, such as upwind schemes, are known to improve finite difference and finite volume methods. In our case, however, there are two additional difficulties. First, we use a CNN to learn the values of the grid functions $\vec{u}^h$ and p^h; and second, we solve the system of equations via a least-squares formulation using a first-order optimizer; cf. chapter 5. It is unclear how modifications such as upwind schemes change the loss landscape; cf. section 3.4, and whether we are able to reliably find a suitable minimum. Thus, individual modifications may not have the expected effects. For this reason, in section 6.4.5 we examine the combination of all the above modifications and study their combined effects.

6.4.1 Higher Order Finite Differences

In all physics-aware models shown so far, we have used centered finite difference approximations of second order to calculate the physics-aware loss. In doing so we have introduced a rather high truncation error of $O\big(h^2\big)$. To reduce this error we can either refine the grid and thus reduce the grid step size h, or we can employ finite difference approximations of higher order. Additionally, when using centered finite-difference approximations, it is possible for non-physical velocity fields to satisfy the discretized divergence-free condition and for non-physical pressure fields to remain undetected. This refers in particular to

oscillations. The inclusion of boundary conditions mitigates this problem locally. Higher order finite difference approximations feature a larger stencil and therefore couple more grid nodes, hopefully damping these oscillations.

In this section we present results for models with centered finite difference approximations of second, fourth, sixth, eighth and tenth order. To perform many computations with limited computational resources in a reasonable amount of time, we again consider models trained on a single geometry. Later, in section 6.4.5, we present results with finite difference approximations of higher order on multiple geometries.

Differences of higher order are not as compact as second-order differences. We pay for the higher accuracy by using additional points for the approximation of the derivatives. While centered differences of second order can be represented as 3×3 matrices, centered differences of fourth-order are already 5×5 matrices; and centered differences of tenth-order, the highest order considered here, are 11×11 matrices. This increases the computational complexity of training a model. However, our CNN already consists of so many convolutions that the larger stencils used to approximate the derivatives do not increase the computation time by much. For example, training a model with second order differences on 1 000 geometry takes about 52 seconds per epoch. This time increases to 57, 58, 61, and 63 seconds for fourth-, sixth-, eighth-, and tenth order differences, respectively. Note that the computation of the residuals is not optimized and we could very possibly improve on these times. Additionally, it is necessary to adjust our boundary treatment so that we do not use nodes that lie outside the geometry for either u or p. To do this, we use low-order stencils close to the edges.

Finally, we have trained up to 10 models per considered difference order on a bifurcation geometry. We have previously seen that our physics-aware approach faces some problems on those geometries; cf. section 6.2.2 and section 6.2.4, especially oscillations in the pressure as well as deviations in the flow field. Therefore, a bifurcation geometry is an excellent test case. To account for training volatility, we trained multiple models per

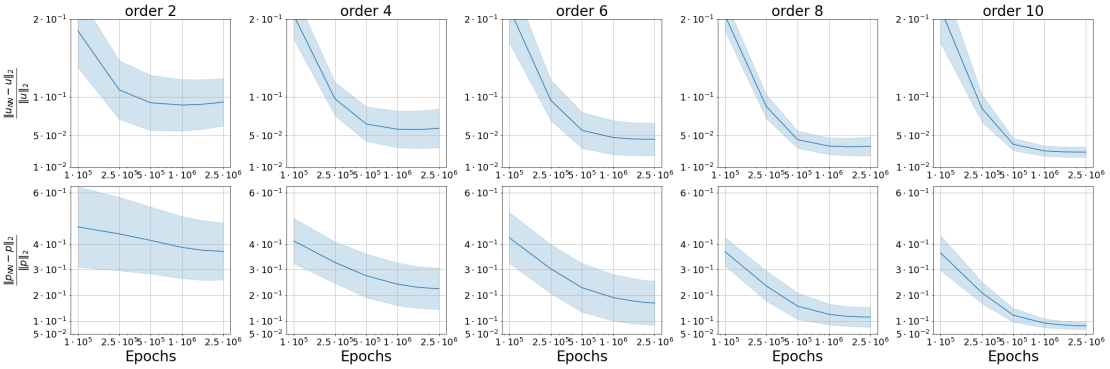
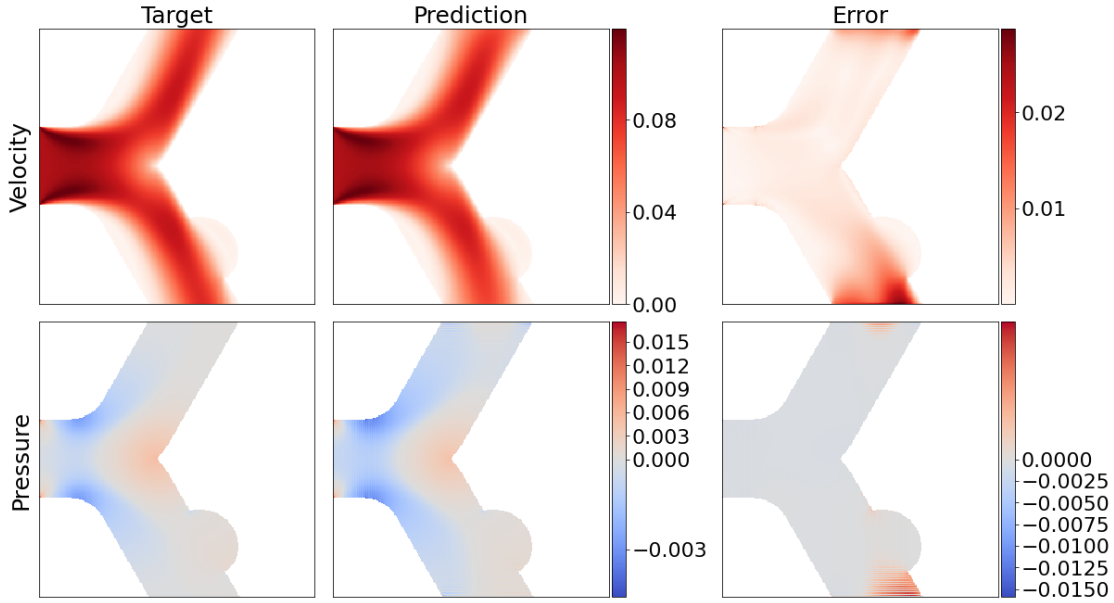


Figure 6.28: Comparison of the histories of the relative errors in u and p over the trained epochs for finite differences of different orders.

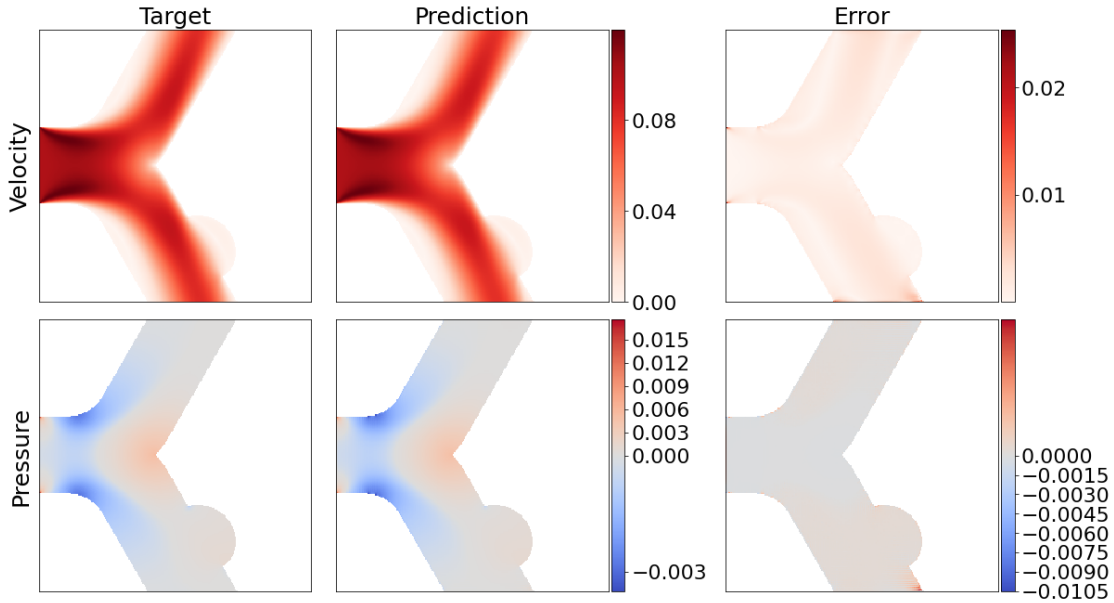
setup.

The results in terms of the relative L_2-errors on the validation data in velocity and pressure are depicted in fig. 6.28. The course of the mean of the errors is plotted on the y-axis against the number of trained epochs on the x-axis. The mean error itself is plotted in dark blue; while a confidence interval, given by the standard deviation, is plotted in a lighter blue. This representation of the error allows us to compare not only the final error after 2 500 000 epochs of training, but also the progression of errors during training. For the final error, we see a clear trend here: the higher the order of the differences used, the lower the final error, both in speed and pressure. From second to tenth-order, the average final error in the velocity is 9.3%, 5.9%, 4.5%, 3.6%, and 2.9% and in pressure 37.2%, 22.7%, 17.0%, 11.6%, and 8.2%. Furthermore, we see a reduction in the volatility of the results for higher order differnces. While for second-order differences, the errors in u vary between 5% and 15% and in p between 20% and 60%, for tenth-order differences the errors for u vary only between 2% and 4%, and for p between 6% and 10%.

In fig. 6.29, two predictions are shown in comparison to the reference data, one for second-order differences and one for tenth-order differences. In the prediction of the model with second-order differences, we see stronger errors near the outflows, quite similar to the



(a) Second order.



(b) Tenth order.

Figure 6.29: Two predictions for a bifurcation geometry, one obtained from a model with second-order differences and one with tenth-order differences.

results in section 6.2.2. Furthermore, we see oscillations in the pressure. The prediction shown is one of the best we obtained with second-order differences, with relative errors of 7.4% in u and 33.3% in p. Thus, the oscillations that we see can be considered weak and they are stronger in the predictions of the other second-order models.

The prediction shown is one of the best, with relative errors of 7.4% in u and 33.3% in p, i.e., the oscillations are not as strong, but can be stronger and are associated with the worse models.

In the prediction of the model with tenth-order differences, we see almost no oscillations. An exception is the lower right corner of the geometry, where slight oscillations occur. Apart from that, the prediction agrees very well with the reference solution, which is also reflected in the relative errors of 2.3% in u and 7.1% in p.

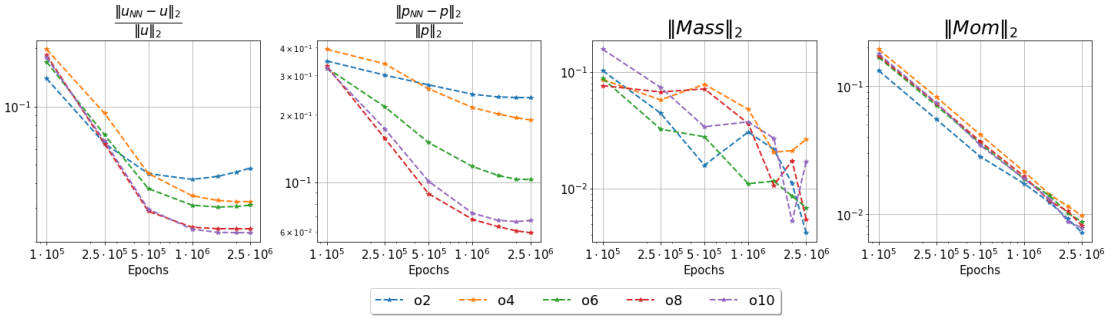


Figure 6.30: History of the relative errors and the norm of the residuals over the number of trained epochs for the best models per difference order from fig. 6.28.

In fig. 6.30 we show the relative errors in u and p and the norms of the residuals of the governing equations for the respective best models per order in comparison. Note that the size of the residuals does not correlate with the relative errors. This is to be expected, however, since the sources of the error remains undetected, e.g. due to the use of centered differences in the case of oscillations. In addition, it is noticeable that the residual of the momentum equation is strictly decreasing, in contrast to the residual of the mass equation. This indicates that the residual of the momentum equation may dominate the

residual of the mass equation, further motivating us to increase the weight of the residual of the mass equation.

In summary, the use of higher order centered differences reduces the error in the predictions. However, this is rather due to the weaker oscillations and not necessarily due to the higher accuracy of the centered differences. Thus, explicitly in the areas where no, or only weak, oscillations occurred, we can hardly detect an improvement. One of the most likely reasons for this is that the performance of the model is not limited by the accuracy of the used stencils, but by the minimization of the loss. We know that training a neural network is a highly complex task; cf. section 3.4, and that it is possible to get stuck in local minima. In the literature, e.g. for PINNs [143], better results were obtained with optimizers like L-BFGS [110] than with Adam [85]. To avoid the higher cost of L-BFGS for the whole training procedure, e.g., a two-step training was recommended in [65], where first the loss was minimized with Adam up to a certain value, and then the model was further trained with L-BFGS. A similar approach might be advantageous in our case. However, the use of the L-BFGS algorithm as an optimizer is not investigated in this work.

6.4.2 Pressure Stabilization

Previously, we have discussed the occurrence of oscillations in the solution of Navier–Stokes equations discretized with finite differences; cf. section 2.2.3. One reason for this is the failure of our chosen discretization, which sonstis of an unstaggered grid configuration and centered differences of the same order for all velocity and pressure terms, to satisfy the inf-sup condition; and another is the inability of centered differences to detect certain oscillations in the pressure. We have also discussed possible solutions, such as the use of a staggered grid configuration. In this section, we investigate how the introduction of a small stabilization term into the divergence-free equation affects the predictions of our models.

We have already introduced this penalty term in equation eq. (2.27) of section section 2.2.3 and show it again here for convenience. For a small parameter ϵ, we modify the divergence-free equation so that it becomes

$$\nabla \cdot \vec{u} = \epsilon \Delta p. \tag{6.1}$$

Other stabilization techniques are possible, and we refer to [95, Sect. 4.1] for an overview. Now, ϵ is a small value that must be chosen. It should not be too large, in order not to distort the solution too much, but also not too small; otherwise pressure oscillations may occur again. For this reason, we test a range of values for ϵ in this section. We examine this stabilization on three channel geometries that are similar to the geometries we considered in section 6.2.1. Thus we refer to these geometries again as the first, second, and third geometries. For these geometries, different flow fields emerge and the maximum velocity ranges from $\approx 6\frac{m}{s}$ to $\approx 13\frac{m}{s}$.

Since the predictions contain errors whose causes are not eliminated by this pressure stabilization alone, a purely quantitative comparison should be treated with caution. In particular, it is advisable to examine the predictions qualitatively and to explicitly consider the pressure oscillations.

We train models for $\epsilon \in \left\{0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\right\}$. In fig. 6.34 we present the relative errors in the form of a boxplot of 4 models for each value of ϵ. In addition, in fig. 6.31, fig. 6.32 and fig. 6.33 we compare the predictions for one representative model per value of ϵ, one figure for each of the three geometries.

From the quantitative error plots, we can draw two immediate conclusions. First, a stabilization parameter of $\epsilon = 10^{-2}$ is too high for all geometries; the prediction errors are very high for all 4 models. Thus, we do not need to consider a higher value of ϵ, since this would only lead to an even stronger bias of the prediction. Second, for the remaining epsilon values, the distributions of error values look quite similar. An exception to this is the stabilization with $\epsilon = 10^{-3}$ for the third geometry. This seems to be already erroneous after evaluation of the quantitative errors. The second conclusion is

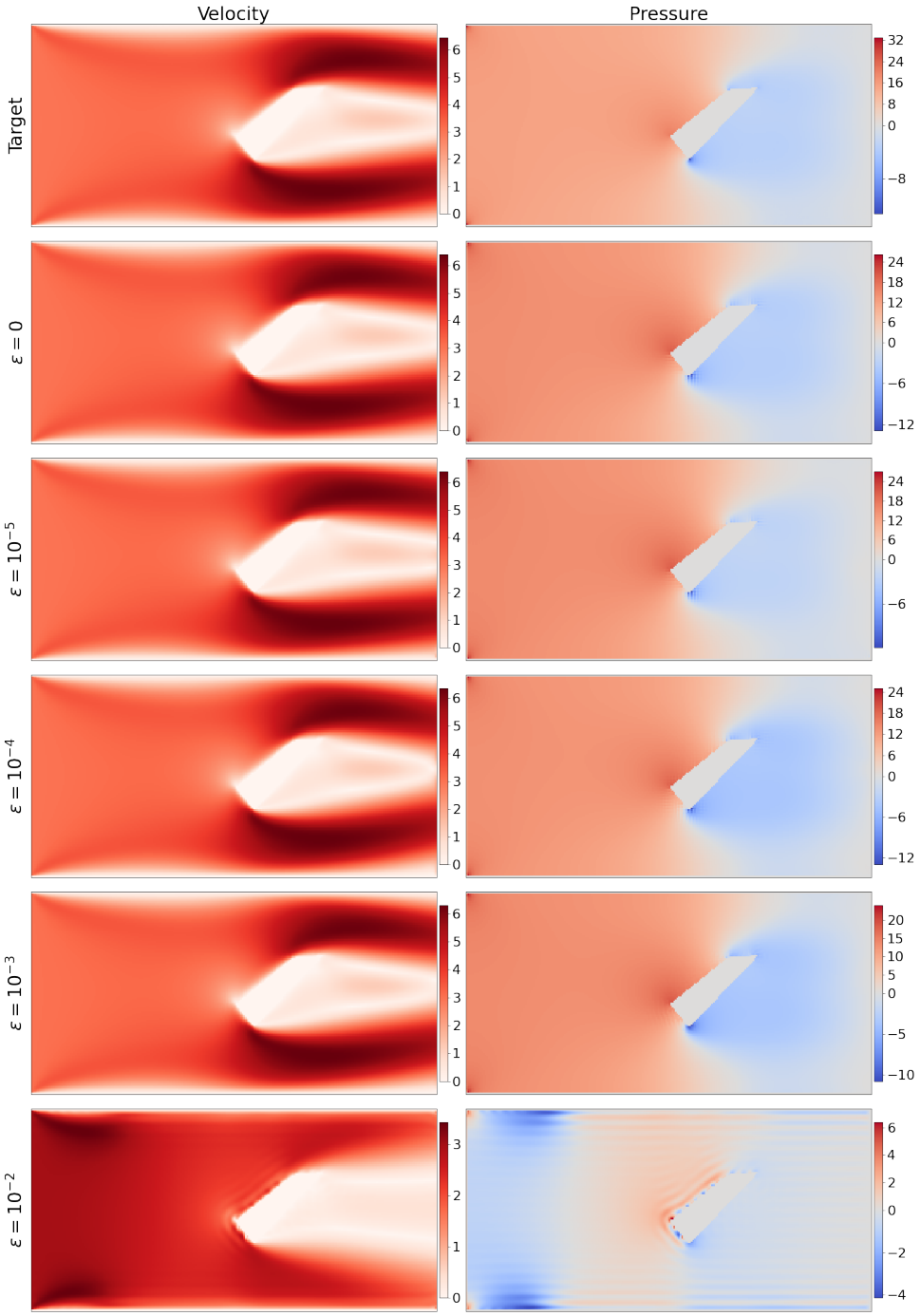


Figure 6.31: Comparison of velocity and pressure for physics-based models trained with different values of ϵ on the first geometry.

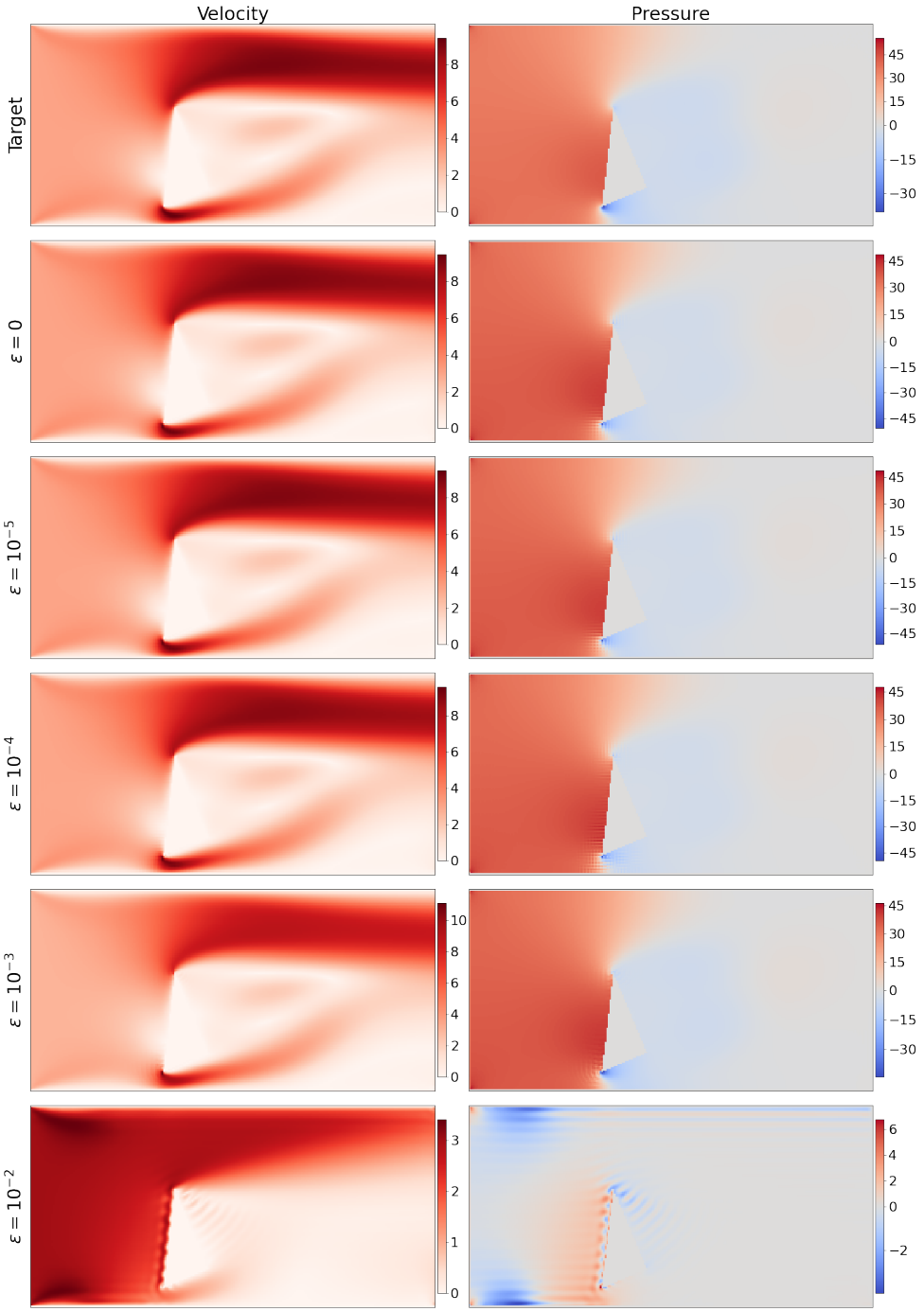


Figure 6.32: Comparison of velocity and pressure for physics-based models trained with different values of ϵ on the second geometry.

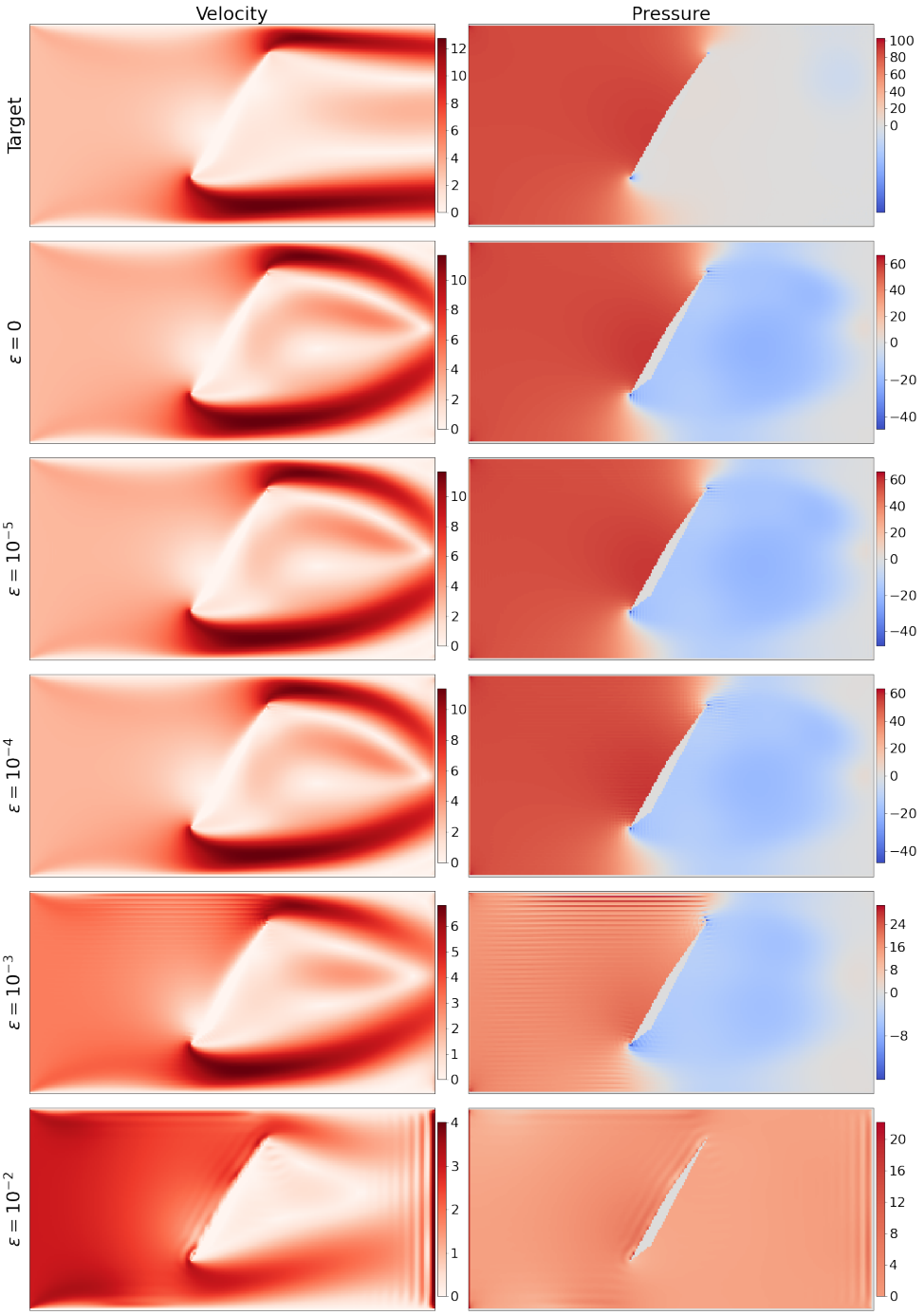
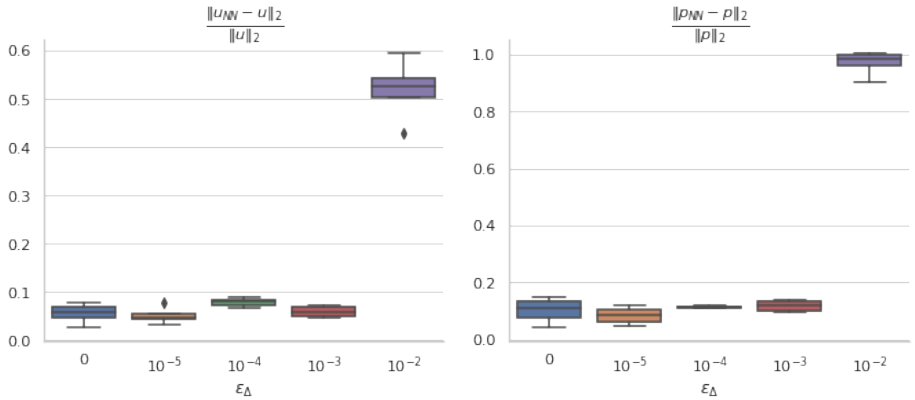
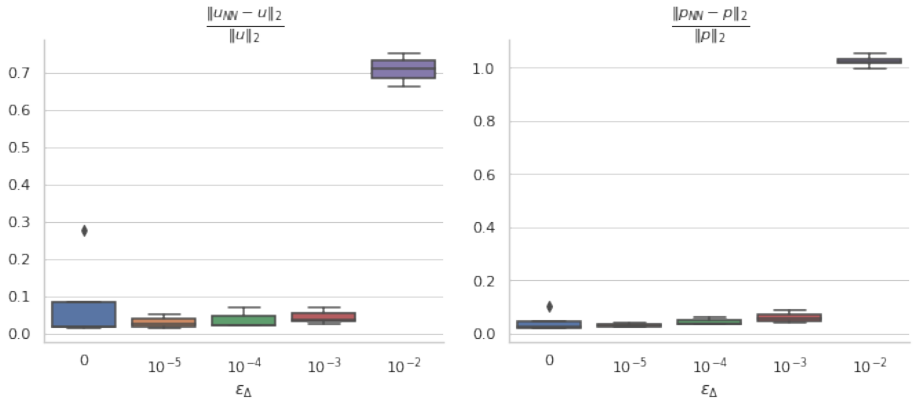


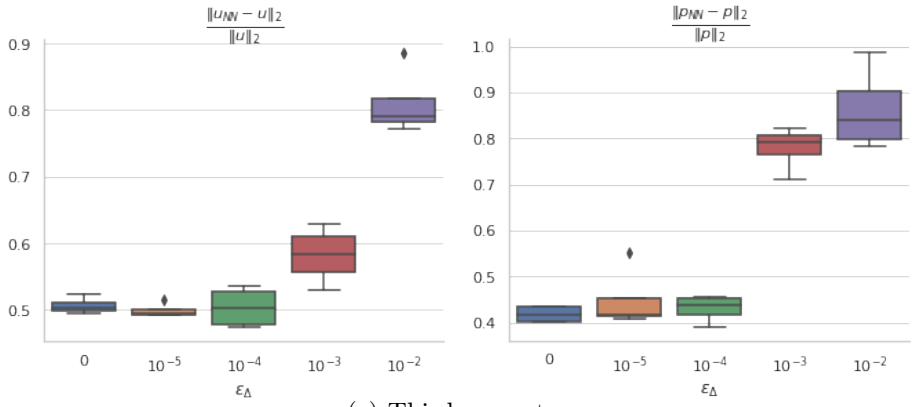
Figure 6.33: Comparison of velocity and pressure for physics-based models trained with different values of ϵ on the third geometry.



(a) First geometry.



(b) Second geometry.



(c) Third geometry.

Figure 6.34: Relative errors of the velocity and pressure for physics-based models trained with varying values of ϵ.

very desirable, since a constant error with an increasing stabilization parameter at least means that the prediction is not falsified by the stabilization. Whether this also means that the pressure fluctuations decrease is not evident from the quantitative errors.

As mentioned above, a purely quantitative evaluation of the results is not conclusive, since the training of the models is also influenced by other effects. Therefore, it is highly recommended to examine the individual predictions qualitatively, at least exemplarily. Thus, in the qualitative comparison of the predictions for the first geometry; cf. fig. 6.31, we see that without stabilization, i.e. $\epsilon = 0$, there are oscillations in the predicted pressure close to the obstacle. These oscillations become weaker and weaker with increasing ϵ. For $\epsilon = 10^{-4}$ they are already much weaker, and for $\epsilon = 10^{-3}$ they are barely visible. A similar effect can be observed in the predictions of the other two geometries; cf. fig. 6.32 and fig. 6.31. Here, however, a stabilization with $\epsilon = 10^{-3}$ already leads to a distortion of the flow. For the third geometry, it is even the case that new, strong oscillations appear in flow and pressure.

Hence, we can say that the stabilization chosen here with a suitable parameter ϵ has a stabilizing effect with respect to the occurrence of oscillations in the predicted pressure. For all three geometries considered, $\epsilon = 10^{-4}$ was found to be the most appropriate parameter. This damped the pressure oscillations without distorting the predicted velocity.

6.4.3 Upwind Schemes

We have observed in previous results that our method has difficulties with higher velocity flow fields; cf. section 6.2.1 and section 6.2.3. We know from our preliminary work in section 2.2.3 that in a finite difference context this problem is related to certain properties of finite difference approximations or discretized difference equations; cf. [187, Chapt. 5] and [123, Sect. 2.3.1]. One way to solve this problem would be to use a finer resolution. However, due to the architecture of a CNN, we cannot refine the mesh locally, so we would have to refine the whole mesh uniformly. This would drastically increase the

computational cost and lead to other problems; see section 6.1.1. Therefore, we turn to the other possible solution, the use of upwind schemes. However, it is not clear how the use of such methods in the computation of the physics-aware loss affects the training and predictions of our models, since we do not directly minimize the residuals as vectors, but use the minimization of the residuals to train a CNN as a surrogate model for the discrete solutions $\vec{u}$ and p. For this reason, we study two upwind schemes, namely first-order upwind in section 6.4.3.1 and hybrid upwind in section 6.4.3.2. Later, we also consider higher-order upwind schemes; see section 6.4.5.

In general, an upwind scheme consists of using finite difference approximations that involve more points on the upwind side. We have provided a mathematical motivation for the use of upwind schemes in section 2.2.3. Often a different, heuristic and physical argument is made, commonly in the context of the finite volume method. This argument states that in a strongly convective flow, the flow on a cell wall is more strongly influenced by the upwind flow than by the downwind flow and that a differencing scheme should take this into account; cf. [187].

6.4.3.1 First-Order Upwind Scheme

The first-order upwind scheme [138] is the simplest upwind scheme available. It simply considers the flow direction and uses one-sided first-order finite difference approximations, i.e., forward or backward, for the derivatives in the convective terms. We still approximate all of the other terms by second-order centered differences. In general, the convective terms of the Navier–Stokes equations are $(\vec{u} \cdot \nabla)\vec{u}$; cf. section 2.1. In two dimensions they are given explicitly by

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} \tag{6.2}$$

in the x-component of the momentum equation and by

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} \tag{6.3}$$

in the y-component of the momentum equation. According to the scheme, we approximate $\frac{\partial u}{\partial x}$ and $\frac{\partial v}{\partial x}$ depending on u and $\frac{\partial u}{\partial y}$ and $\frac{\partial v}{\partial y}$ depending on v. In particular, if $u > 0$ we use first-order backward differences

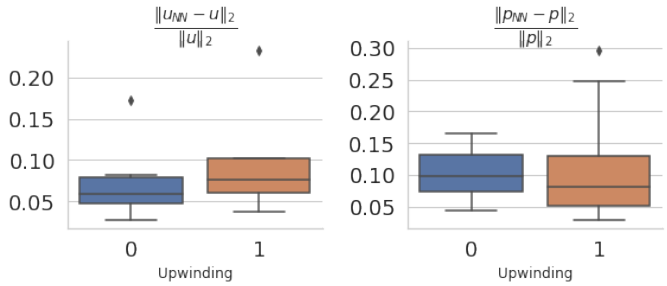
$$\frac{\partial u}{\partial x} \approx \frac{u_{i,j} - u_{i-1,j}}{h} \quad \frac{\partial v}{\partial x} \approx \frac{v_{i,j} - v_{i-1,j}}{h}$$

and if $u < 0$ we use first-order forward differences

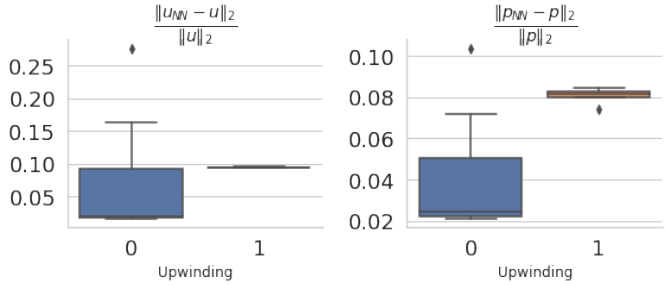
$$\frac{\partial u}{\partial x} \approx \frac{u_{i+1,j} - u_{i,j}}{h} \quad \frac{\partial v}{\partial x} \approx \frac{v_{i+1,j} - v_{i,j}}{h}.$$

We use the same procedure accordingly for $\frac{\partial u}{\partial y}$ and $\frac{\partial v}{\partial y}$ depending on v. Note that with the one-sided differences that are used here, only the values at the upwind nodes are being utilized. More importantly, the upwind differencing scheme is bounded for all values of u and v. This is a property that the central differencing scheme does not share. However, the upwind scheme has its drawbacks. Namely that of accuracy. One-sided first-order differences are only first-order accurate, i.e., the truncation error is $O(h)$ instead of $O(h^2)$ as for centered differences. Another major drawback is, that the upwind scheme introduces false diffusion when the flow is not aligned with the grid orientation; cf. [187]. The stencils used are smaller than centered stencils, but because we use both forward and backward approximations, the total stencil to be considered is exactly the same size as that of second order centered differences. Consequently, we expect the first-order upwind scheme to give better results in convection-dominated flows, but worse results due to lower accuracy and false diffusion in areas where the flow is not or only weakly convection-dominated.

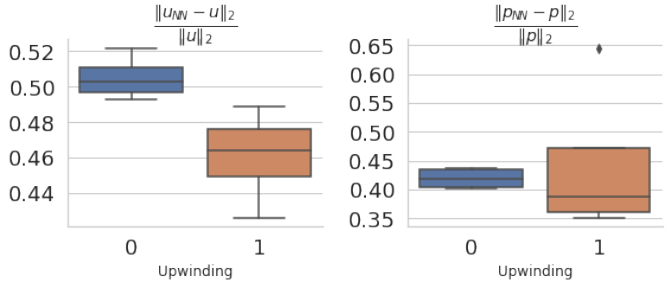
We examine the effect of the upwind scheme on the same three channel geometries that we considered for the pressure stabilization; cf. section 6.4.2. We will again refer to these geometries as the first, second, and third geometries. For these geometries, we obtain different flow fields with maximum velocities ranging from approx. $6\frac{m}{s}$ to approx. $13\frac{m}{s}$. For each geometry we train up to eight models each with and without the upwind scheme for the convective terms and plot the relative errors as a boxplot in fig. 6.35, as in the



(a) First geometry.



(b) Second geometry.



(c) Third geometry.

Figure 6.35: Relative errors of velocity and pressure for the physics-based models with and without first-order upwind.

previous section. Additionally, in fig. 6.36 we compare the predictions for one model each with and without upwind on the second geometry. Using the upwind scheme for the convective terms seems to worsen the models' predictions for the first geometry, as can be seen from the boxplots. This is not surprising, since the problems our physics-aware

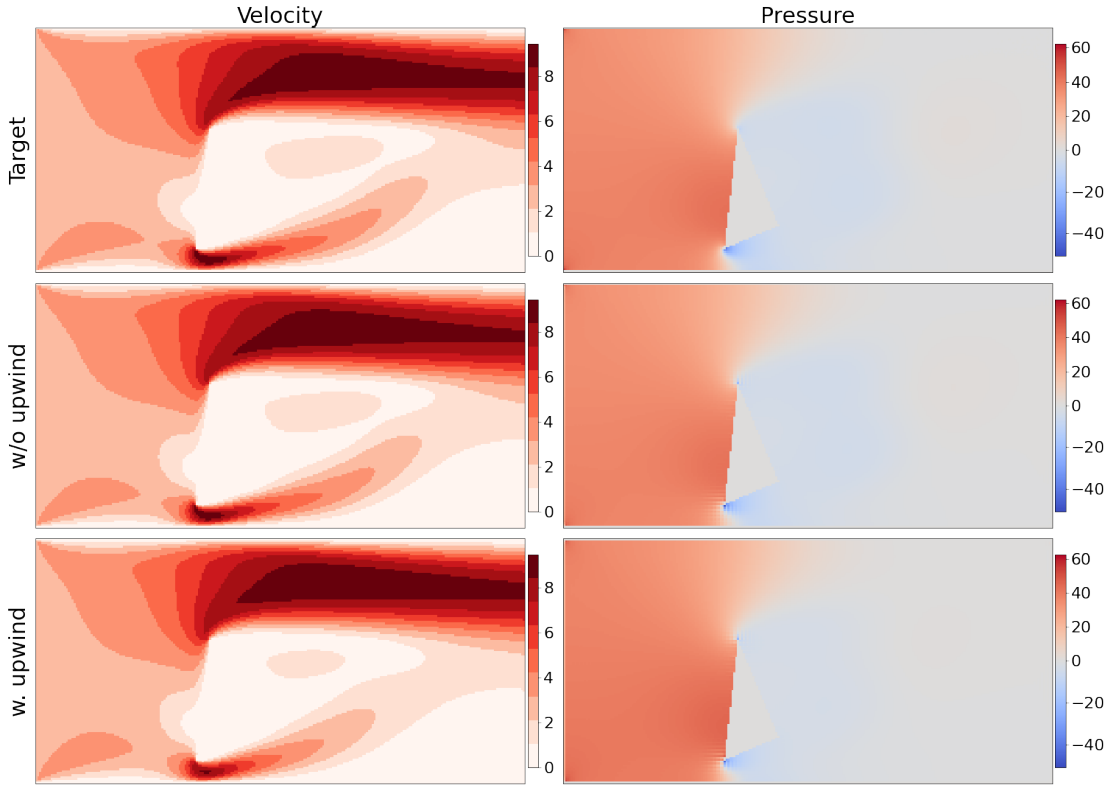


Figure 6.36: Comparison of velocity and pressure for physics-based models trained with and without first-order upwind on the second geometry.

CNNs have with the first geometry are rather related to oscillations in the prediction than to convective-dominant flow. The lower accuracy of the first-order upwind scheme as well as the false diffusion added hurt our models capabilities in this case. For the second geometry, the errors have increased using the upwind scheme. We examine possible reasons on two example predictions for this geometry in detail later. However, let us note that the errors of the predictions for the second geometry of the models without upwind are surprisingly low. For the third geometry, we see that no meaningful prediction has been learned with or without the use of the upwind scheme. As we noted in section 6.2.1, this is not unexpected since we most likely need to employ more than just a first-order

upwind scheme to obtain a meaningful prediction for this geometry. However, we note that there is some, albeit small, improvement for the third geometry.

Let us take a closer look at the predictions shown in fig. 6.36. Here, we compare the best prediction without upwind with a relative error in velocity of 1.5% to the best prediction with upwind with a relative error in velocity of 8.3%. Note that the prediction without upwind is an exceptionally good fit, as we have previously reported higher errors for comparable geometry; see section 6.2.1.2. There are at least two observations to be made here. First, in the predictions of the upwind model, there is some false diffusion in the lower part of the geometry just behind the obstacle. The flow in this region is not aligned with the grid orientation. For such a flow, the first-order upwind scheme introduces false diffusion, so this is expected. In fact, especially for wall-bounded flows close to the boundaries, this false diffusion can cause the prediction to be inaccurate; cf. [123, Sect. 2.3.1]. Second, the velocity in the upper part of the geometry behind the obstacle is slightly overestimated in the upwind model prediction. In contrast, predictions generally underestimate the maximum velocity when centered differences are used. It is possible that the strong false diffusion in the lower part of the geometry influences the flow behavior in the upper part of the geometry, leading to this behavior.

In summary, the use of the first-order upwind scheme does not improve the predictions of the models on any of the three geometries considered. In principle, improvements could have been obtained for the second and third geometries. For the second geometry, the improvements are most likely not obtained due to the false diffusion that occurred. For the third geometry, the difficulties of our approach are manifold and it was not expected that a simple first-order upwind scheme alone would lead to a meaningful prediction. However, the previously discussed difficulties in training the model still remain, and using an upwind scheme only addresses one of them. In the next section, we use a more sophisticated upwind scheme, the so-called hybrid upwind scheme.

6.4.3.2 Hybrid Upwind Scheme

The hybrid upwind scheme [172] is a combination of the first-order upwind and the central differencing scheme. Depending on the cell-Reynolds number Re_h (Peclet number in thermodynamics) the differencing scheme is decided locally. For low values, i.e. $|Re_h| < 2$, the second-order accurate central difference scheme is used, and for higher values, i.e. $|Re_h| > 2$, the upwind scheme is used, which is less accurate but accounts for transportiveness. Thus, this scheme alleviates some of the problems of the first-order upwind scheme in diffusion-dominated flow, while retaining its advantages in convection-dominated flow.

We recall the definition of the cell Reynolds number from eq. (2.31) and state it again

$$Re_h = \frac{Uh}{\nu}, \tag{6.4}$$

where U is a constant, h the grid step size, and ν the kinematic viscosity. We slightly modify this definition to obtain two metrics by which we can decide whether to switch to the upwind scheme in the x- or in the y-direction. Thus, we define at a grid node the horizontal cell-Reynolds number

$$Re_{hx} = \frac{uh}{\nu} \tag{6.5}$$

and the vertical cell-Reynolds number,

$$Re_{hy} = \frac{vh}{\nu}. \tag{6.6}$$

where u and v are the x- and y- velocity at that grid node. We then decide whether to use first-order upwind differences instead of centered differences based on these two metrics. In particular, if $|Re_{hx}| > 2$ we use centered differences

$$\frac{\partial u}{\partial x} \approx \frac{u_{i+1,j} - u_{i-1,j}}{h}, \qquad \frac{\partial v}{\partial x} \approx \frac{v_{i,j} - v_{i-1,j}}{h},$$

if $0 < Re_{hx} < 2$ we use first-order backward differences

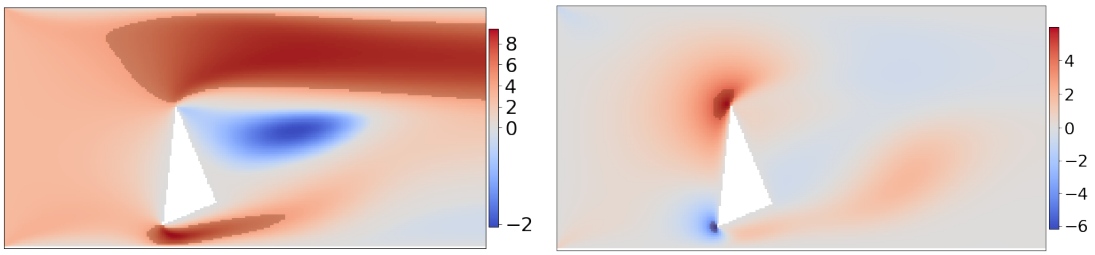
$$\frac{\partial u}{\partial x} \approx \frac{u_{i,j} - u_{i-1,j}}{h}, \qquad \frac{\partial v}{\partial x} \approx \frac{v_{i,j} - v_{i-1,j}}{h},$$

and if $-2 < Re_{hx} < 0$ we use first-order forward differences

$$\frac{\partial u}{\partial x} \approx \frac{u_{i+1,j} - u_{i,j}}{h}, \qquad \frac{\partial v}{\partial x} \approx \frac{v_{i+1,j} - v_{i,j}}{h}.$$

We again use the same procedure accordingly for $\frac{\partial u}{\partial y}$ and $\frac{\partial v}{\partial y}$ depending on Re_{hy}.

For a fixed discretization with constant viscosity, we can again choose the schemes depending on u and v. For the discretization of the domain considered here, the corresponding velocity thresholds are about $\pm 4.25 \frac{m}{s}$. Keep in mind that we can raise or lower these thresholds if we choose to do so. Increasing the thresholds causes the central scheme to be applied in more nodes, while decreasing them causes the upwind scheme to be applied more.

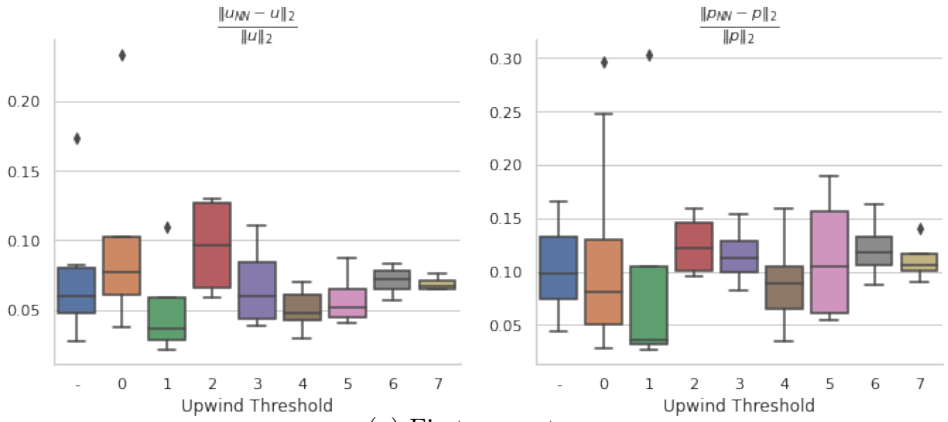


(a) x-velocity (b) y-velocity

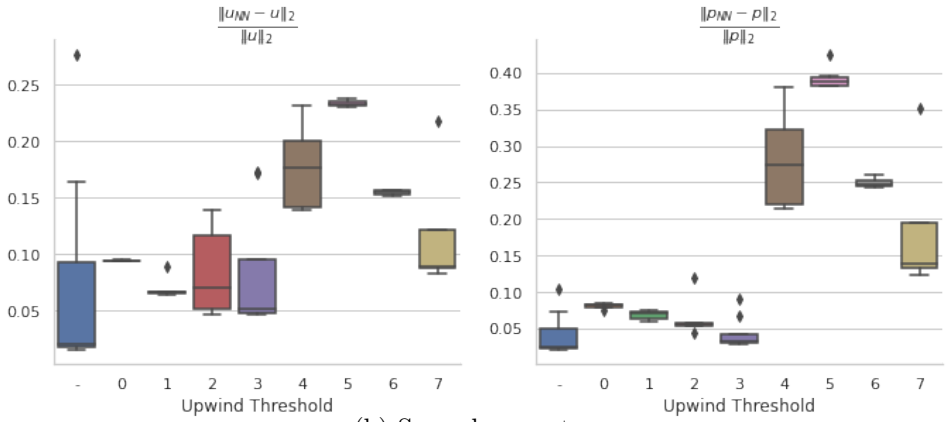
Figure 6.37: The reference flow-field for the second geometry. Areas where the absolute cell Reynolds number is greater than 2 are highlighted.

Using the second geometry as an example, we show in fig. 6.37 the x- and y-components of the velocity of the reference solution, highlighting the regions where the absolute cell Reynolds number is greater than 2 or rather where the respective component of the velocity is greater than 4.25 in absolute terms. Here, the first-order upwind scheme is applied in the hybrid scheme with a threshold of 4.25.

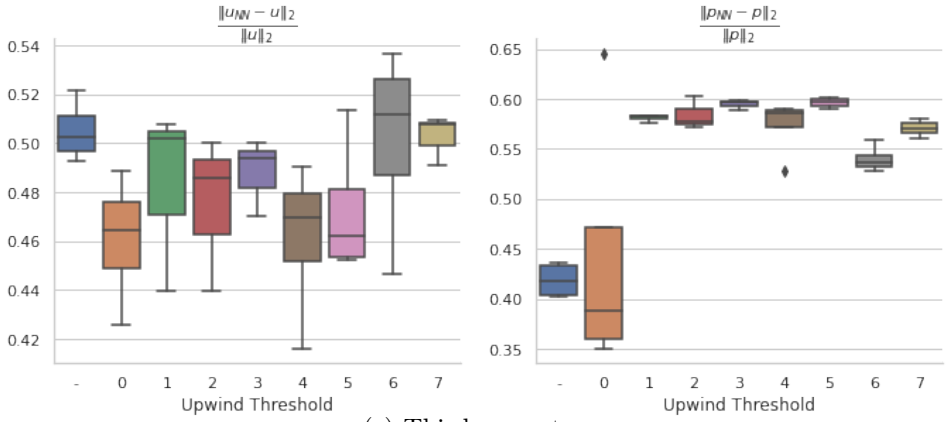
In the best case, we expect our physics-aware CNNs to learn better predictions with the hybrid scheme than with the central scheme, or at least better than with the first-order scheme. In general, however, the hybrid scheme inherits the weaknesses of the first-order



(a) First geometry.



(b) Second geometry.



(c) Third geometry.

Figure 6.38: Relative errors of velocity and pressure for the physics-based models using the central scheme (blue) and hybrid scheme with varying threshold.

or upwind scheme, especially when the upwind scheme is used in many nodes.

We again examine the effect of this scheme on the three channel geometries we considered for the first-order upwind scheme; cf. section 6.4.3.1. In fig. 6.38 we present results for models that use the hybrid scheme with various values for the thresholds in comparison to models that use the central scheme. Note that the hybrid scheme with a threshold of 0 is essentially the first-order upwind scheme.

The results are mixed and must be interpreted with caution. For the first geometry; cf. fig. 6.38(a), we see a good improvement using the hybrid scheme with a velocity threshold of 1, both compared to the first-order upwind scheme and the central scheme. If we now increase the velocity threshold, we would actually expect a further improvement in the results, since the central scheme is used in more nodes where the cell-Reynolds number is less than 2. However, this is not the case. For all other thresholds we see worse results. Notice that the errors for thresholds 2 to 4 again exhibit a decreasing trend. Above a velocity threshold of 4.25, we expect increasingly similar behavior to the central scheme. This expectation is mostly met, at least the errors are similar to those we get with the central system, with surprisingly less variation at higher thresholds. For the first geometry, we compare the predictions of individual selected models in figs. 6.39 and 6.40.

At this point, it is worth reiterating that training a physics-aware CNN is equivalent to solving a high-dimensional nonlinear system of equations with the added difficulty of using a CNN to predict the solution, which is an extremely challenging task. Thus, if we know in the context of finite differences that a certain variation of the methodology leads to the discrete solution being closer to the analytical solution, it does not follow that using that variation of the methodology in our context of a CNN will also lead to better results. This is because the probability of getting stuck in a local minimum is very likely to increase due to the increased complexity of the problem added by using a CNN.

For the second geometry; cf. fig. 6.38(b), we again see good improvement using the hybrid scheme with a velocity threshold of 1 compared to the first-order upwind scheme,

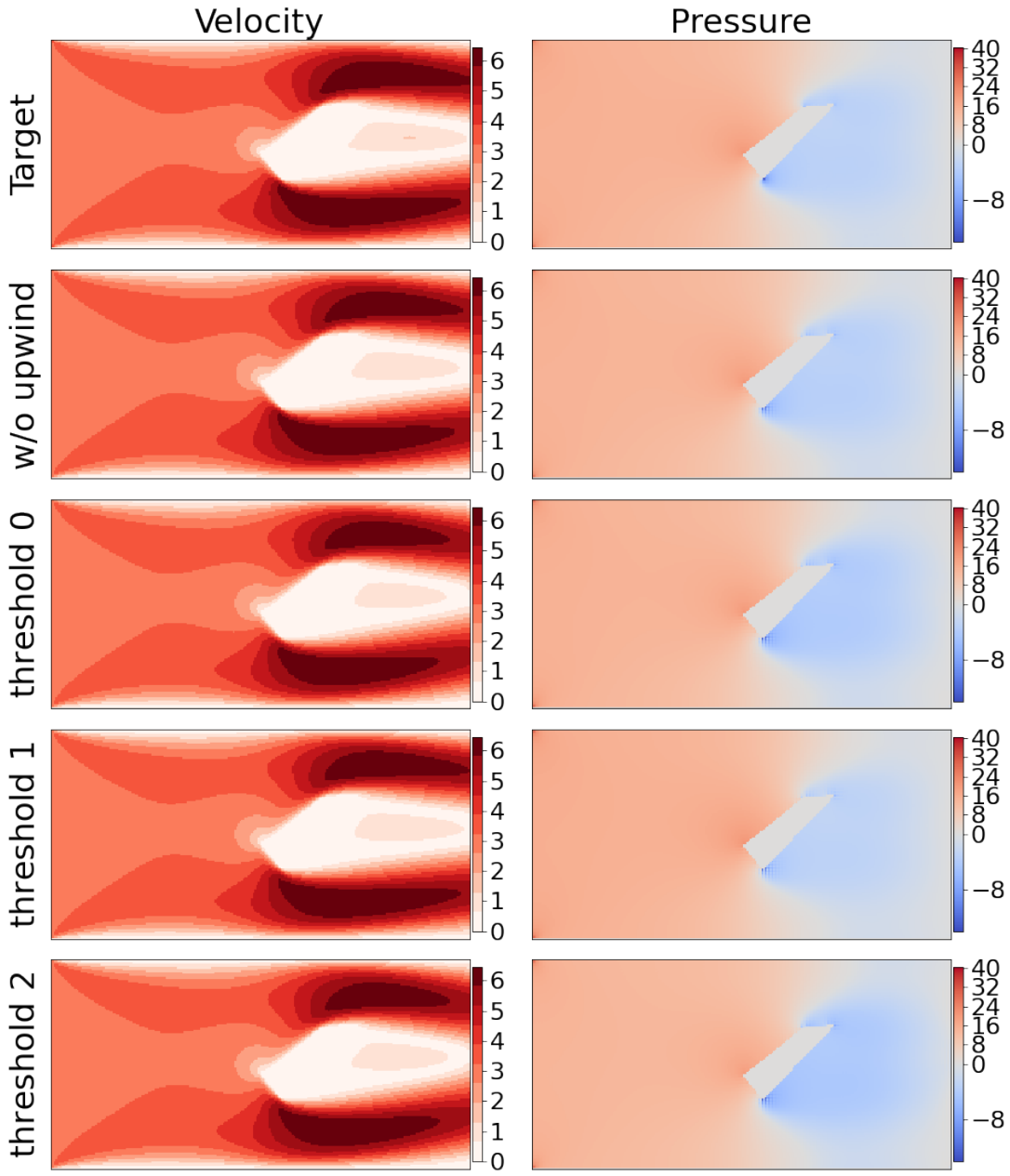


Figure 6.39: Comparison of velocity and pressure for physics-based models trained with and without hybrid upwind on the first geometry, first part.

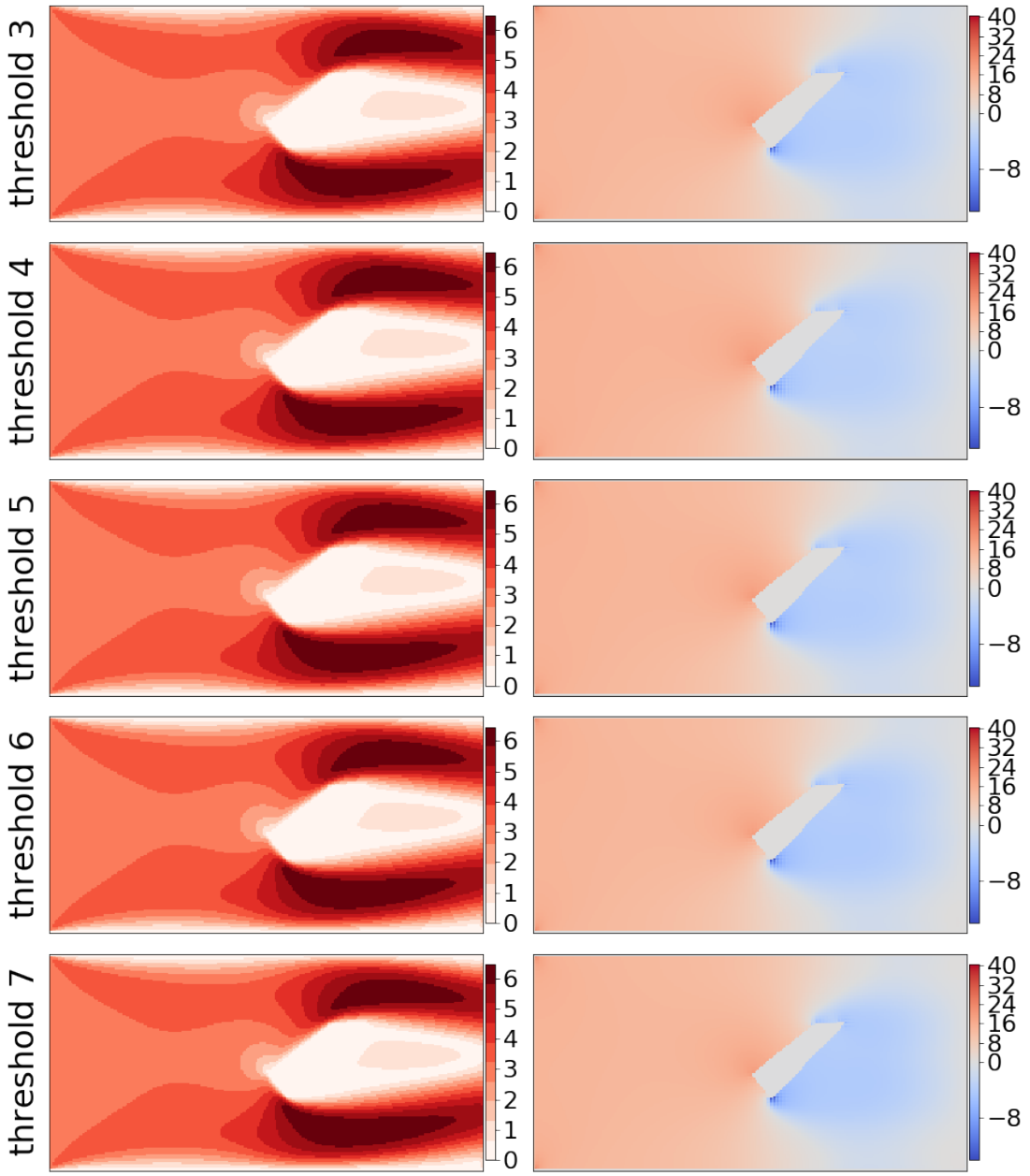


Figure 6.40: Comparison of velocity and pressure for physics-based models trained with and without hybrid upwind for different thresholds on the first geometry, second part.

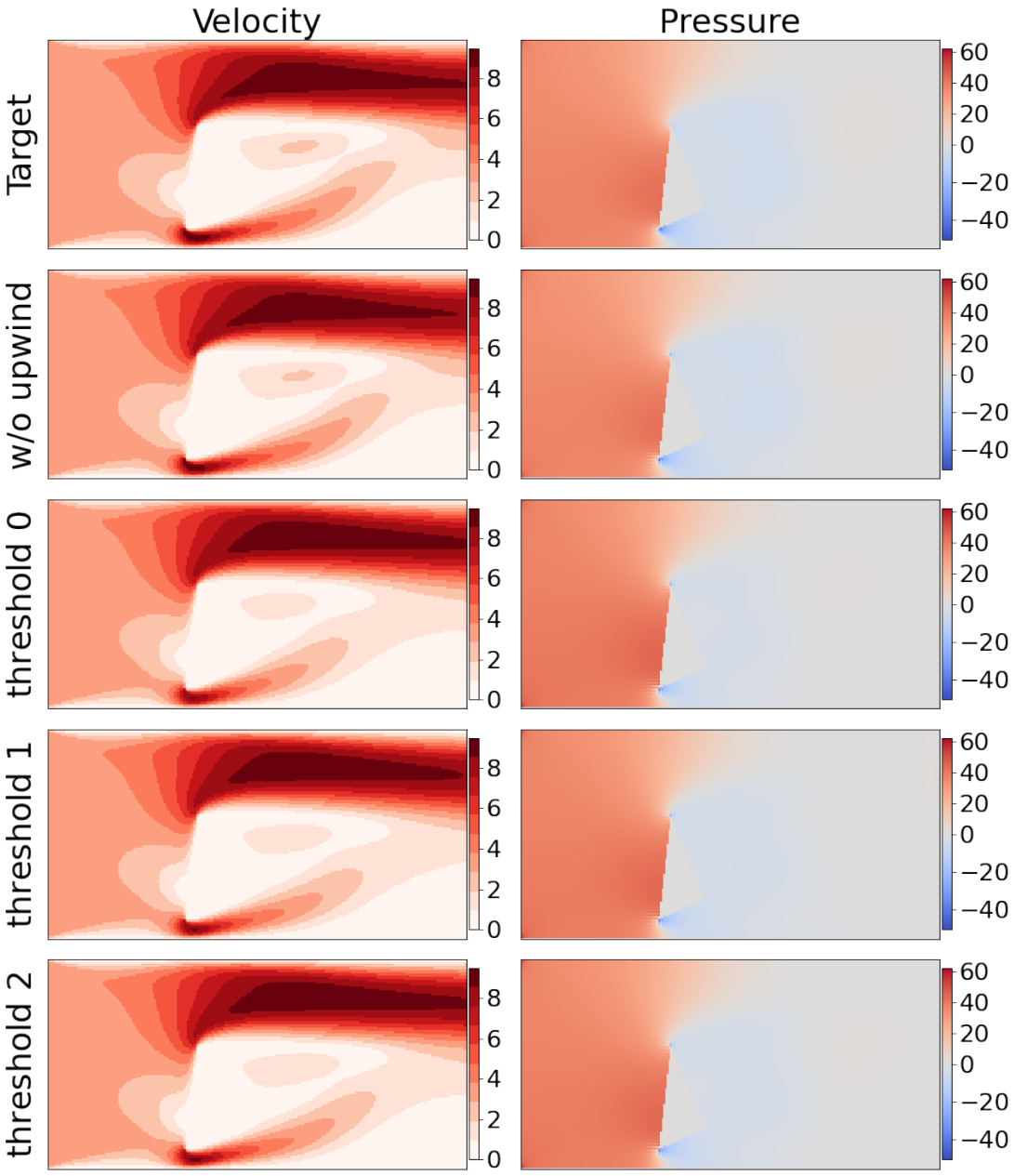


Figure 6.41: Comparison of velocity and pressure for physics-based models trained with and without hybrid upwind on the second geometry, first part.

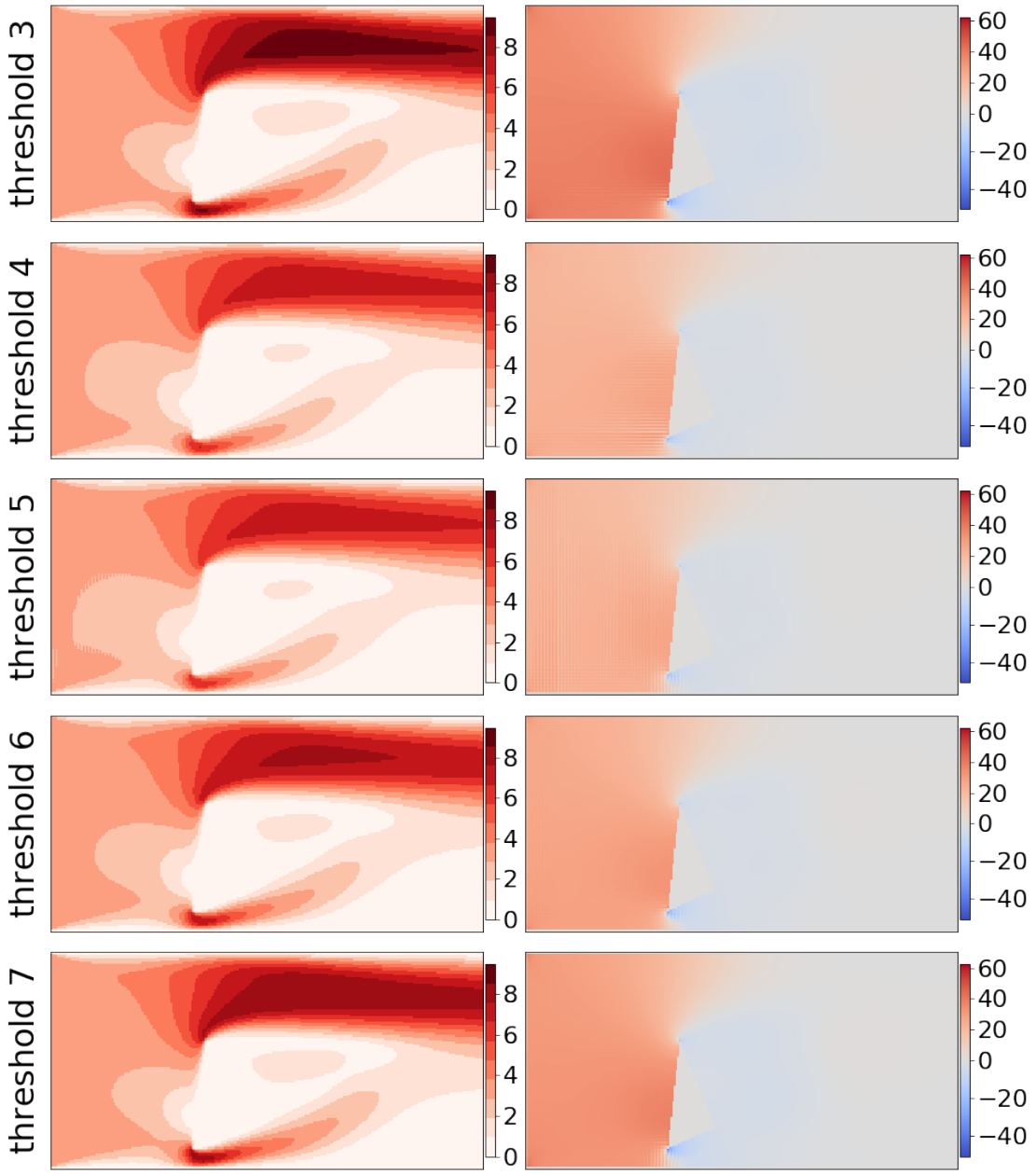


Figure 6.42: Comparison of velocity and pressure for physics-based models trained with and without hybrid upwind for different thresholds on the second geometry, second part.

but not compared to the central scheme. As noted above, the predictions for this geometry without upwind are exceptionally good, and arguably we cannot expect much improvement. In fact, we expect less to no degradation. However, this is not what the results tell us. While there is still some improvement with thresholds of 2 and 3, there is a sharp increase in errors for both velocity and pressure with thresholds of 4 and above. This increase in error is due to the increased and reliable occurrence of oscillations in pressure and, in some cases, velocity; see fig. 6.42. The causes for the occurrence of these oscillations, which did not occur, for example, when the first-order upwind scheme was used, are not clearly discernible, and we refrain from making any untenable assumptions at this point.

For the third geometry; cf. fig. 6.38(c), there is no significant improvement in the prediction error for any velocity threshold. Therefore, we will not further analyze the predictions of any model for this geometry with respect to the hybrid upwind scheme. Note, however, that here too there are strong and consistently occurring oscillations in the predicted pressure.

In summary, some of the results confirm the hypothesis that the hybrid scheme can give better results than the central scheme. In the case of the first geometry, even better results could be obtained with the hybrid scheme than with the central scheme. However, the use of the hybrid scheme seems to potentially introduce instabilities into the training. At least for the second and third geometry, increased and consistent pressure oscillations occurred above a certain velocity threshold, which strongly falsified the learned prediction. Even with an improved upwind scheme, the hybrid scheme, the problems of our approach with geometries that cause a more convection-dominated flow could not be solved. In the following section, we test a final modification, increasing the weighting of the mass residual in the loss function, before examining in section 6.4.5 what the effects are of combinations of the considered modifications.

6.4.4 Weighting of the Loss Terms

In this section we investigate the effect of assigning a higher weight to the mass residual in the loss function. Let us recall the loss function for a single geometry, first presented in eq. (5.25) of chapter 5,

$$\underset{\Psi}{\operatorname{argmin}} \left(\omega_{\text{Mom}} \| N\left(\vec{u}_{NN}\left(I_g\right)\right) + G\, p_{NN}(I_g) \|_2^2 + \omega_{\text{Mass}} \| D\vec{u}_{NN}(I_g) \|_2^2 \right). \tag{6.7}$$

Here, ω_{Mom} and ω_{Mass} are the weights for the two loss terms, the momentum residual and the mass residual. As mentioned earlier, the mass equation is also called the divergence-free condition or constraint.

Since the weights of the CNN are randomly initialized, the prediction does not initially satisfy the divergence-free equation. During training, we minimize the sum of the squared residuals eq. (6.7), and with equal weights $\omega_{\text{Mom}} = \omega_{\text{Mass}} = 1$ it is very possible that the learned prediction satisfies the momentum equation more than the mass equation. While it is true that a valid solution must satisfy both the mass equation and the momentum equation, we know that the mass equation is more of a constraint that limits the space of valid solutions. In addition, we use a form of the Navier–Stokes equations, or rather the momentum equation, in whose derivation $\nabla \cdot \vec{u} = 0$ was explicitly assumed and used for simplification; cf. [53] and our discussion of this in section 2.1. Thus, although we are equally interested in solving the momentum equations, it may be advantageous to restrict the search space to velocity fields that satisfy the divergence-free condition. Since we cannot easily modify the architecture of our CNNs to ensure that their predictions are always divergence-free, we will attempt to achieve a similar result by increasing the weight ω_{Mass} of the mass residual loss term in the loss function. Another approach could be to use a different form of the momentum equation where $\nabla \cdot \vec{u} = 0$ is not assumed in its derivation. However, this is not investigated in this work.

To investigate the effect of higher weighting of the mass residual, we train four models each for a weight of 1, 10, 100, and 1 000 on the three previously studied geometries;

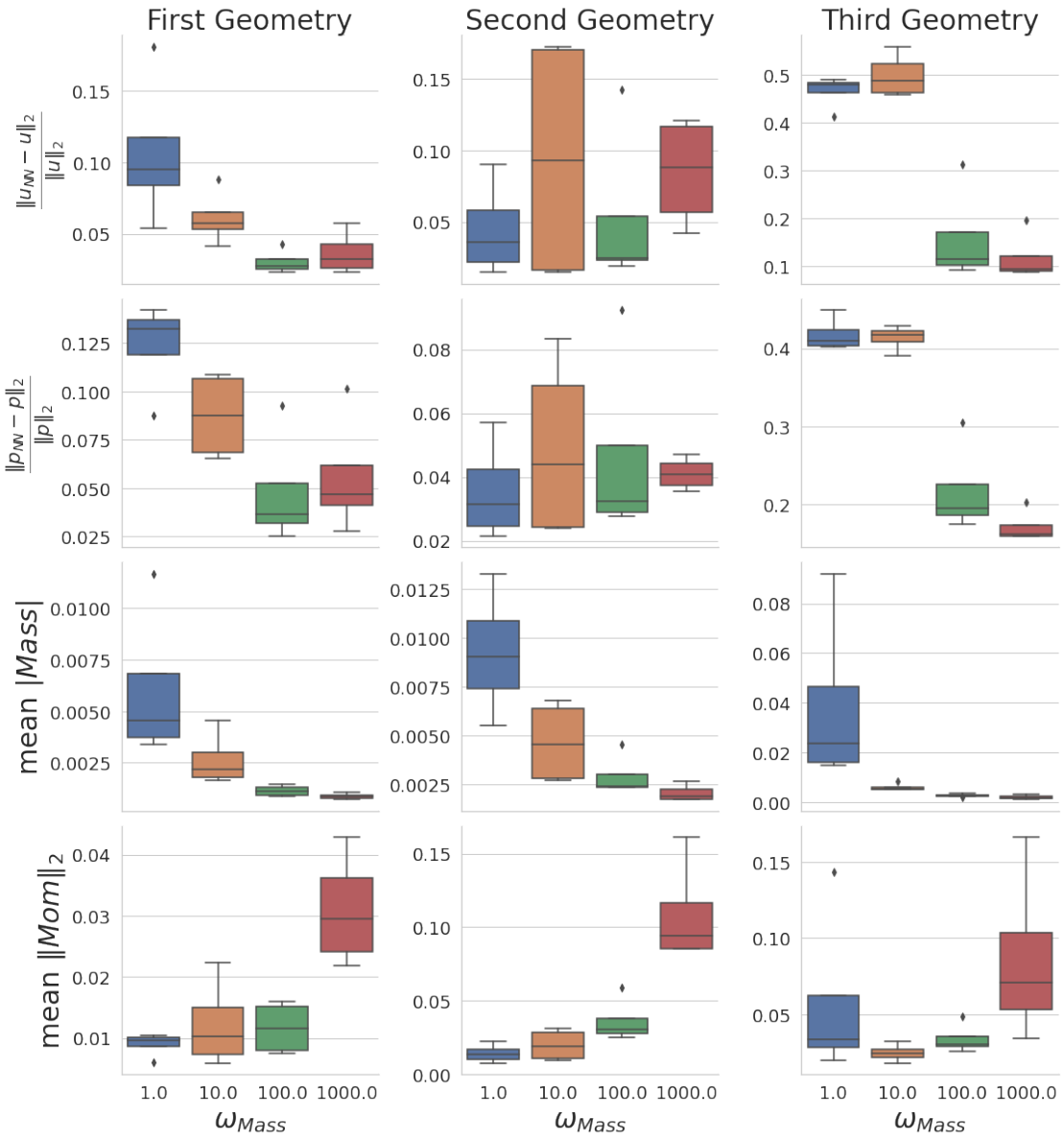


Figure 6.43: Relative errors of velocity and pressure for the physics-based models for different values of the weight ω_{Mass} of the mass-residual.

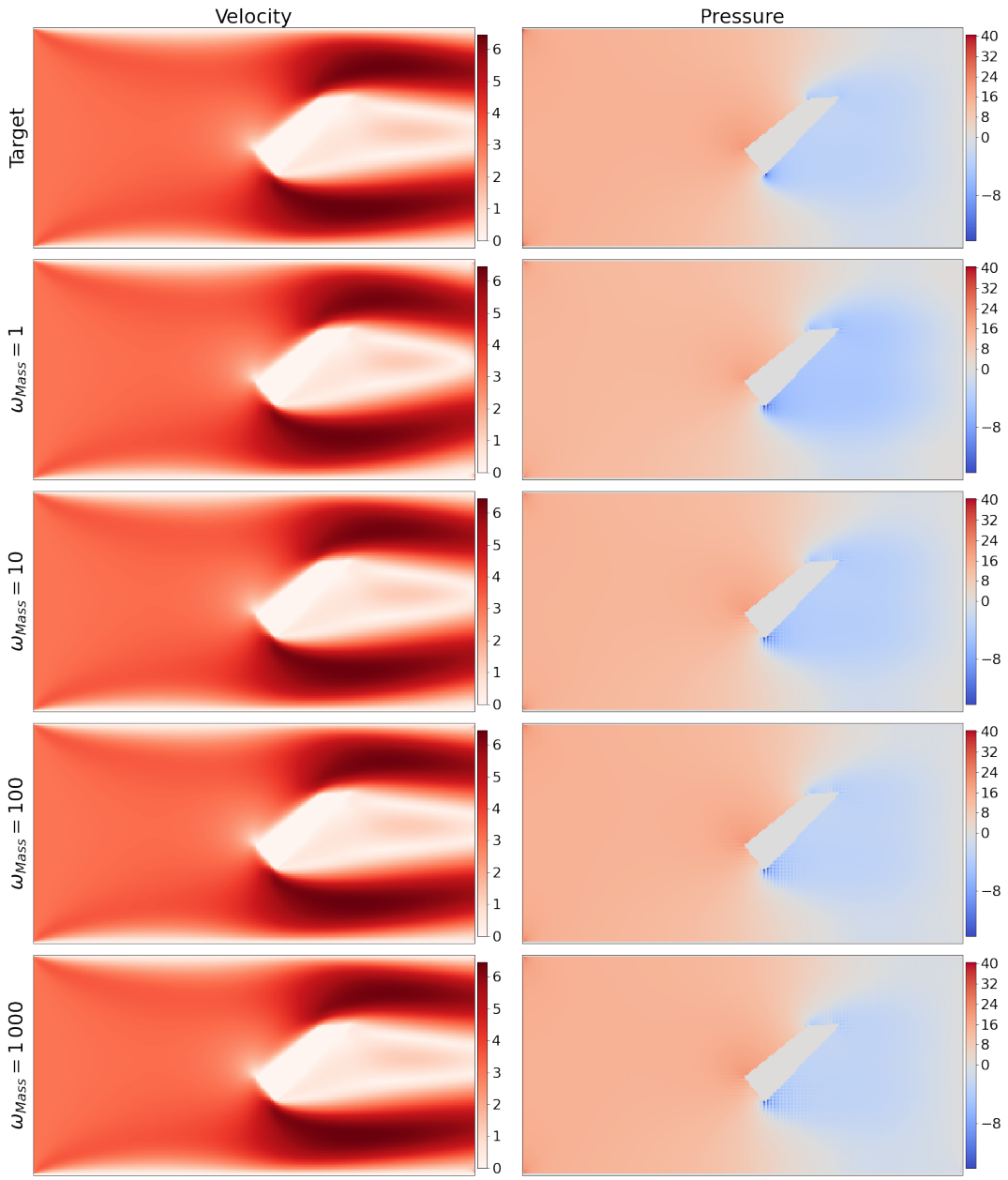


Figure 6.44: Comparison of velocity and pressure for physics-based models trained with various values of ω_{Mass} on the first geometry.

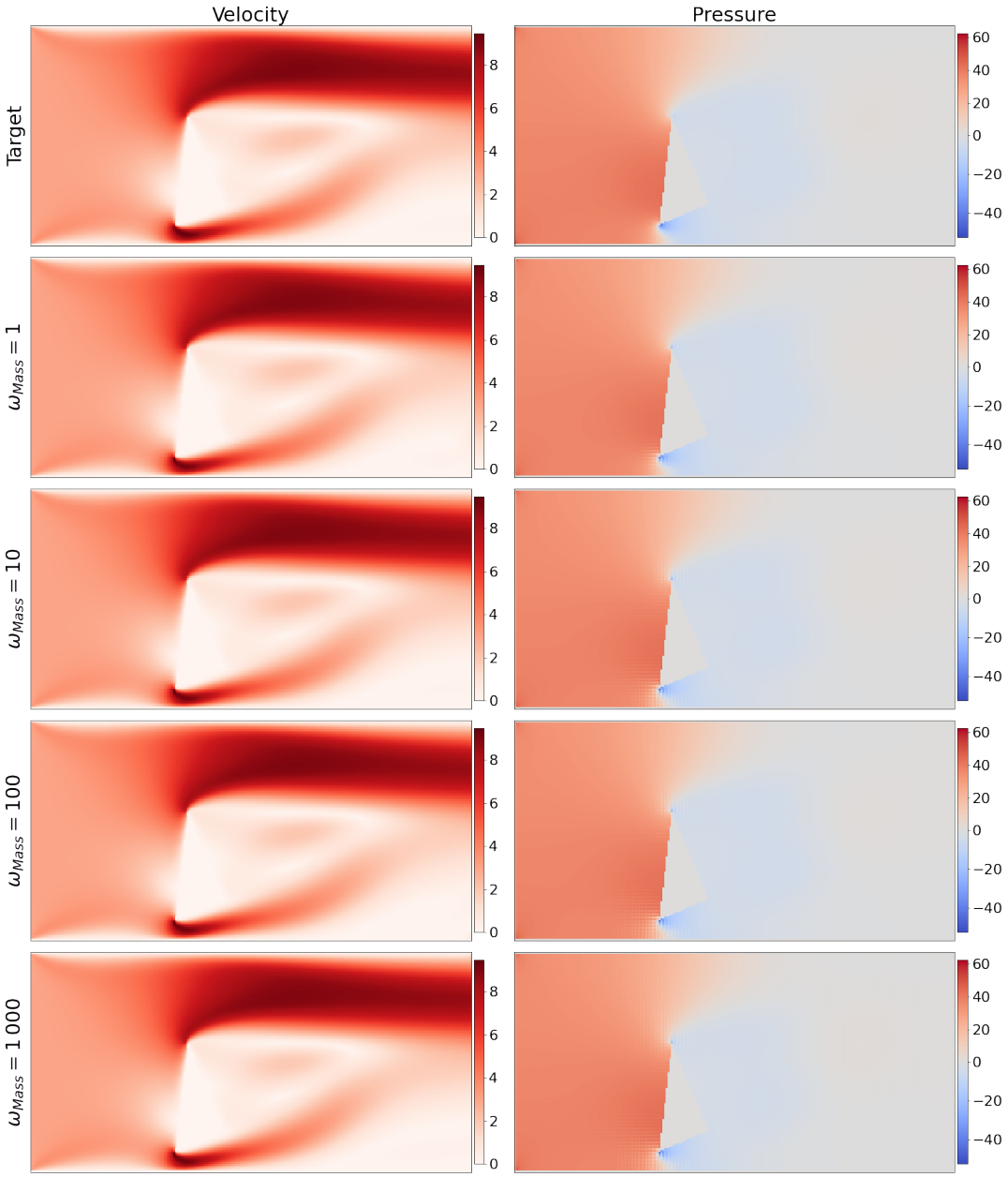


Figure 6.45: Comparison of velocity and pressure for physics-based models trained with various values of ω_{Mass} on the second geometry.

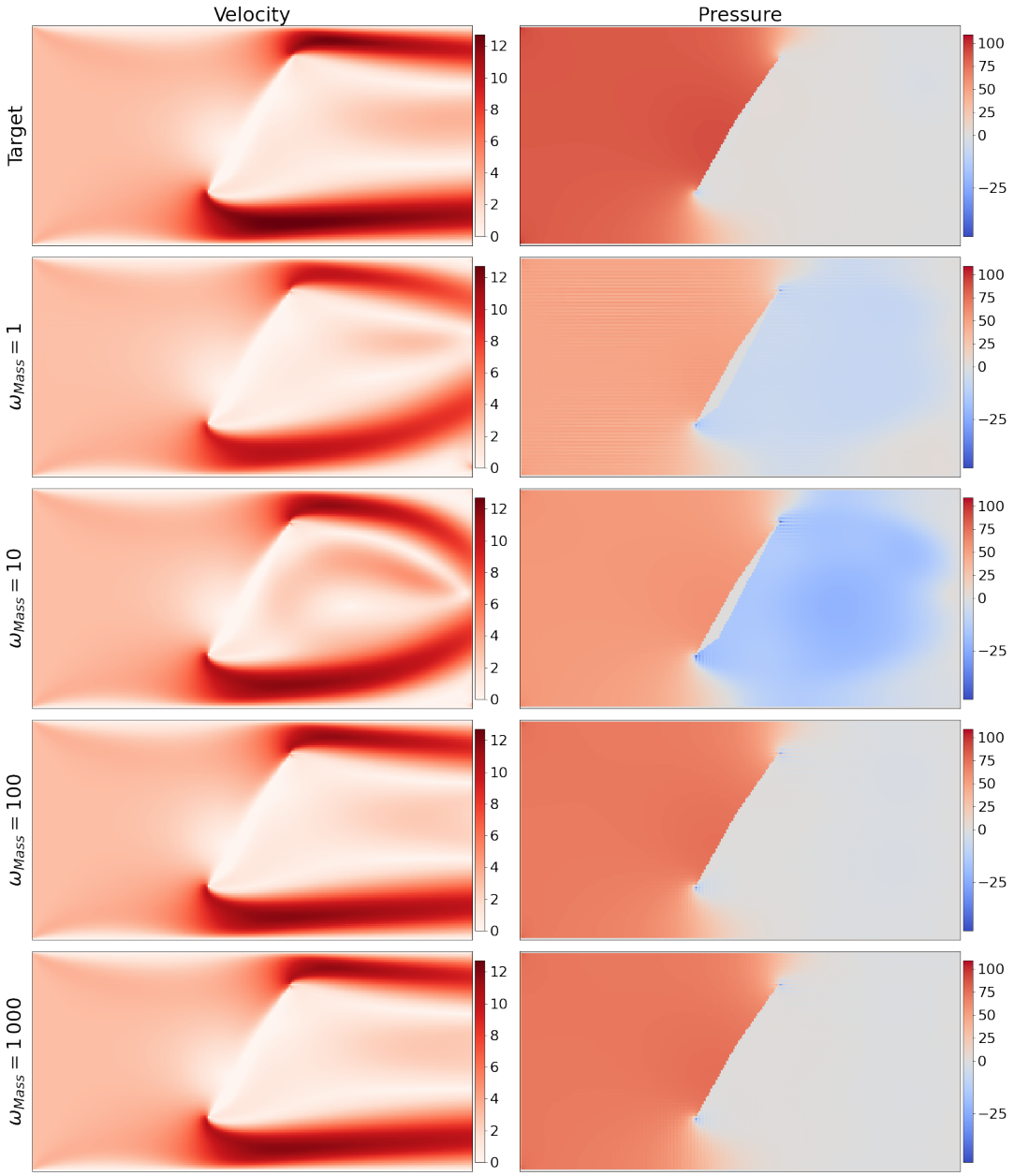


Figure 6.46: Comparison of velocity and pressure for physics-based models trained with various values of ω_{Mass} on the third geometry.

cf. section 6.4.2 and section 6.4.3. In fig. 6.43 we show the relative errors in u and p, as well as the values of the absolute mass residual |Mass| averaged over all inner pixels for a geometry. and the norm of the moment residual $\|\text{Mom}\|_2$. For the first and third geometries, the errors in u and p decrease with increasing weight. For the first geometry, the errors increase again slightly with a weight of 1 000, while for the third geometry they continue to decrease even at this weight. For the second geometry, compared to the standard case of a weight of 1, the errors do not decrease for a weight of 10. However, the errors are significantly lower on average for a weight of 100, especially in the velocity. For a weight of 1 000 the errors increase again.
The results for the third geometry are particularly interesting. For a weight of 1 and 10, we get errors whose magnitudes suggest that no meaningful flow has been learned. For the higher weights, 100 and 1 000, the velocity errors are reduced to only 10%. This large reduction in errors suggests that with a higher residual mass weight, meaningful flow is learned for this geometry as well.
However, we compare not only the errors of the predictions for velocity and pressure, but also the residuals averaged over the inner pixels. As expected, a stronger weighting of the mass residual leads to a steady decrease of the averaged absolute divergence. At the same time the values of the momentum residual increase. Up to a weight of 100, however, the mass residual decreases more than the momentum residual increases, especially in the first and third geometry. It is also important to keep in mind that a low momentum residual paired with a high mass residual is in no way indicative of a properly learned flow field; cf. section 2.1. Thus, a slightly higher momentum residual paired with a lower mass residual is preferable to a lower momentum residual paired with a higher mass residual. Overall, it can be seen that a weight of 100 brings an improvement in all geometries, and even significantly in the third one. In the following, the individual predictions are considered in detail.

In figs. 6.44, 6.45, and 6.46, we compare the predictions of individual selected models,

one for each weight, for the three geometries, i.e., the first, second, and third, respectively. For the first geometry, see fig. 6.44, the errors of the four predictions shown are 5.4%, 4.2%, 2.3%, and 2.7% for velocity and 8.8%, 6.5%, 2.5%, and 2.8% for pressure. While our approach faces no major problems with geometries like this in principle, the quality of the predictions is even better for models trained with a higher weighting of the mass residual. However, it can be seen in the predictions that with a higher weighting of the mass residual, the oscillations in the pressure assume a broader extent. While for $\omega_{\text{Mass}} = 1$ the oscillations are still confined to the immediate vicinity of the sharp corners of the obstacle, they become more widespread as the value of ω_{Mass} increases. It is possible that these oscillations cause the errors to start increasing again at the weight $\omega_{\text{Mass}} = 1\,000$. Moreover, too high a weight ω_{Mass} in principle causes the momentum residual in the loss function to become comparably irrelevant small, and the learned predictions fail to satisfy the momentum equation much at all. Note that for all predictions, errors in the prediction of the velocity occur at the upper and lower corners of the outflow edge.

For the second geometry, see fig. 6.45, the errors of the four predictions shown are 2.4%, 1.4%, 1.8%, and 4.2% for velocity and 2.6%, 2.4%, 2.7%, and 3.5% for pressure. The quality of the predictions is also improved for this geometry by a higher weighting of the mass residual, although the learned prediction for $\omega_{\text{Mass}} = 1$ is already a very good fit, as noted in the previous sections. The amplification of the oscillations in p for higher weights ω_{Mass}, already observed for the first geometry, is also evident in the predictions of the second geometry.

For the third geometry, see fig. 6.46, the errors of the four predictions shown are 41.4%, 45.9%, 9.1%, and 8.6% for velocity and 44.9%, 41.4%, 17.4%, and 16.3% for pressure. The predictions for the weights $\omega_{\text{Mass}} = 1$ and $\omega_{\text{Mass}} = 10$ show an incorrect flow pattern in the region behind the obstacle. Also, the learned pressure does not match the reference solution. In addition, the pressure for weight $\omega_{\text{Mass}} = 1$ shows strong pressure variations throughout the geometry. With a higher weight of the mass equation, the predicted

flow pattern is much more similar to the reference solution, and the pressure is also qualitatively more correct. However, the velocity errors are still comparatively high at 9.1% and 8.6%. However, this geometry is a special case, which is a challenge not only for our method. At the same time, in the pixel representation of the geometry, the obstacle at the upper end is not continuous, but has a small gap, which is not present in the real geometry; see section 6.2.1.3. Errors as large as 9% are probably about the best we can expect on this geometry, given all the other sources of errors such as the rasterization of the geometry and under-resolution.

In summary, a higher weighting of the mass residual in the loss function leads to a strong improvement in the quality of the predictions. It is important not to set the weight ω_{Mass} too high, otherwise the momentum residual will not be considered enough in the training. However, at higher weights ω_{Mass}, there are increased pressure oscillations near the corners of the geometry boundaries. It is interesting to see if these oscillations can possibly be reduced through higher order finite differences or, for example, through pressure stabilization. In the next section, we will examine the effect of combining the modifications considered so far.

6.4.5 Combination of Modifications - Single Geometries

In the previous sections, we tested four changes in the calculation of residuals. First, we used higher-order finite differences and found that pressure fluctuations were less likely to occur in this case. However, despite the lower truncation error, the overall accuracy of the predictions did not increase; cf. section 6.4.1. Second, we introduced a penalty term to stabilize the pressure in section 6.4.2 which prevents or at least reduces pressure fluctuations if the parameter ϵ is chosen correctly. In section 6.4.3 we tested upwind schemes to address the difficulties our approach has with higher velocity flow fields. Here the effectiveness was limited. Finally, in section 6.4.4 we tested a higher weighting of the mass residual in the loss function and found significant improvements for all three

geometries considered.

In this section, we examine the effects of combinations of these modifications. We consider second and sixth-order finite differences, pressure stabilization with $\epsilon = 10^{-4}$, the hybrid upwind scheme with threshold 4.25, and mass residual weights ω_{Mass} of 1, 100, and 1 000. Due to the large number of possible combinations, available computational resources, and time constraints, we train only two models per combination. We show the relative errors of velocity and pressure for second-order finite differences in table 6.3 and for sixth-order finite differences in table 6.4. In each row of the table, the first three entries indicate which combination of the three modifications is involved. The first entry indicates which weight was used for the mass residual, the second whether the hybrid upwind scheme was used, and the third whether pressure stabilization was used. The following entries are the relative errors in velocity and pressure for the three geometries.

First, we discuss the second-order finite difference results listed in table 6.3. The results for models with only one modification have already been discussed in the previous sections. Thus, for the second-order finite differences only the combinations of a higher weight for the mass residual and pressure stabilization, a higher weight for the mass residual and hybrid upwind, and finally all three modifications combined need to be discussed.

The combination of higher weighting of the mass residual and pressure stabilization leads to improvements compared to the application of pressure stabilization alone. However, the results with a higher residual mass weighting without pressure stabilization are better for all three geometries. So the combination of these two changes does not result in a significant improvement. With pressure stabilization, however, we introduce a penalty term into the mass equation. If we now increase the weight of the mass residual in the loss function, we may have to decrease its parameter ϵ so that the perturbation of the mass equation does not obscure the solution of the momentum equation.

The combination of a higher mass residual weight and the hybrid upwind scheme also yields improvements over using upwind alone on all three geometries. In this case, the

Geometry			First		Second		Third	
ω_{Mass}	uw.	ϵ	$\frac{\|u_{NN}-u\|}{\|u\|}$ (%)	$\frac{\|p_{NN}-p\|}{\|p\|}$ (%)	$\frac{\|u_{NN}-u\|}{\|u\|}$ (%)	$\frac{\|p_{NN}-p\|}{\|p\|}$ (%)	$\frac{\|u_{NN}-u\|}{\|u\|}$ (%)	$\frac{\|p_{NN}-p\|}{\|p\|}$ (%)
1	0	0	9.35	12.95	2.43	2.55	41.36	44.91
1	0	0	5.36	8.77	4.72	3.75	47.95	41.53
100	0	0	2.60	3.87	2.42	2.92	10.62	18.93
100	0	0	2.87	3.37	1.87	2.77	9.15	17.44
1 000	0	0	2.30	4.53	4.22	3.54	8.61	16.30
1 000	0	0	3.31	4.54	6.10	4.71	9.10	15.92
1	0	10^{-4}	4.89	7.43	2.28	3.86	51.57	40.86
1	0	10^{-4}	8.87	10.71	1.88	3.70	47.93	46.12
100	0	10^{-4}	2.74	4.14	3.20	4.15	34.47	33.61
100	0	10^{-4}	2.84	4.47	4.56	10.60	24.42	26.87
1 000	0	10^{-4}	4.36	7.54	10.31	9.59	10.28	16.60
1 000	0	10^{-4}	4.01	6.77	9.59	13.23	13.12	18.58
1	1	0	6.66	11.21	19.45	31.12	49.64	55.63
1	1	0	8.41	10.83	19.33	31.98	50.20	55.32
100	1	0	3.10	4.14	4.26	3.92	10.14	18.36
100	1	0	2.76	3.29	4.47	3.17	10.14	18.08
1 000	1	0	7.59	23.82	13.98	6.89	9.14	15.24
1 000	1	0	3.04	4.50	7.22	4.39	10.01	16.20
1	1	10^{-4}	4.75	7.68	24.13	39.52	47.25	54.02
1	1	10^{-4}	7.68	10.29	43.87	31.85	47.73	54.67
100	1	10^{-4}	3.48	5.35	4.73	5.21	36.85	39.17
100	1	10^{-4}	4.64	6.54	6.93	12.47	10.27	15.48
1 000	1	10^{-4}	5.19	9.95	8.20	9.18	39.09	38.12
1 000	1	10^{-4}	5.59	15.47	6.40	4.52	15.35	20.90

Table 6.3: Relative errors of velocity and pressure for the physics-based models for combinations of the previously discussed modifications with second-order differences.

errors for $\omega_{\text{Mass}} = 100$ and hybrid upwind on the first and third geometries are similar to the errors for $\omega_{\text{Mass}} = 100$ without hybrid upwind. On the other hand, on the second geometry for $\omega_{\text{Mass}} = 100$ we see a degradation with upwind compared to without upwind. Therefore, we cannot say that the combination of a higher weighting of the residual mass and the hybrid upwind scheme brings improvements.

The combination of the three modifications does not bring any improvement for any of the considered values of ω_{Mass}. In direct comparison, all other combinations produce better results.

In summary, the largest impact is from the weighting of the residual mass. Across all combinations considered, we obtained the best results here for $\omega_{\text{Mass}} = 100$. This is consistent with the observations made in the previous sections. A higher weighting of the mass residual in combination with one of the other two modifications with second-order differences did not lead to better results.

Then, we discuss the sixth-order finite difference results listed in table 6.4. The results of this table are new and have not been discuss in any previous section, so all results need to be discussed here.

First, we consider the results for a higher weighting of the mass residual with sixth-order differences. The results are similar to those with second-order differences. For a higher value of ω_{Mass} we achieve better results on the first and third geometries, while we do not see any degradation on the second geometry. As before, we achieve the best results with $\omega_{\text{Mass}} = 100$, while we already see degradations for $\omega_{\text{Mass}} = 1\,000$ for the first and second geometry. In general, we obtain lower pressure errors with sixth-order differences on the first and second geometries than with second-order differences. According to the observations made in section 6.4.1, this can be explained by the damping effect higher order stencils have on pressure oscillations. However, the errors for the third geometry for a weight of $\omega_{\text{Mass}} = 100$ are higher than for the second-order differences. Nevertheless, the best result so far is 7.86% relative velocity error on the third geometry with $\omega_{\text{Mass}} = 1\,000$.

Geometry			First		Second		Third	
ω_{Mass}	uw.	ϵ	$\frac{\|u_{NN}-u\|}{\|u\|}$ (%)	$\frac{\|p_{NN}-p\|}{\|p\|}$ (%)	$\frac{\|u_{NN}-u\|}{\|u\|}$ (%)	$\frac{\|p_{NN}-p\|}{\|p\|}$ (%)	$\frac{\|u_{NN}-u\|}{\|u\|}$ (%)	$\frac{\|p_{NN}-p\|}{\|p\|}$ (%)
1	0	0	8.08	10.35	47.41	28.44	55.91	52.62
1	0	0	7.43	8.61	1.86	1.34	52.32	36.13
100	0	0	2.80	2.32	1.62	1.96	20.79	21.13
100	0	0	2.42	2.73	1.69	1.85	19.69	21.34
1 000	0	0	5.43	8.01	3.91	2.00	21.97	22.60
1 000	0	0	3.66	5.89	5.19	2.18	7.86	14.09
1	0	10^{-4}	8.42	9.94	13.28	10.60	46.26	44.12
1	0	10^{-4}	4.58	8.41	5.74	5.06	50.80	44.34
100	0	10^{-4}	3.21	5.04	2.72	4.17	8.77	15.67
100	0	10^{-4}	2.86	3.43	2.59	5.01	8.88	16.29
1 000	0	10^{-4}	2.88	3.21	8.05	6.84	8.74	15.12
1 000	0	10^{-4}	4.37	8.74	8.75	4.98	9.10	17.52
1	1	0	10.14	10.30	65.64	33.47	39.87	53.19
1	1	0	8.73	11.63	6.65	8.29	79.51	97.61
100	1	0	3.00	4.93	2.39	2.95	34.15	33.22
100	1	0	2.43	3.18	2.61	3.00	35.48	33.04
1 000	1	0	3.69	5.85	5.95	3.54	11.23	14.57
1 000	1	0	4.03	6.39	19.32	5.78	8.76	13.53
1	1	10^{-4}	17.67	24.37	10.48	16.69	45.83	57.22
1	1	10^{-4}	15.13	34.70	11.07	17.90	47.02	50.58
100	1	10^{-4}	2.90	2.52	2.84	2.62	20.47	26.39
100	1	10^{-4}	2.47	2.85	2.73	3.99	18.99	24.65
1 000	1	10^{-4}	3.28	3.78	10.17	8.34	8.59	16.42
1 000	1	10^{-4}	2.75	5.15	9.96	7.88	10.51	18.90

Table 6.4: Relative errors of velocity and pressure for the physics-based models for combinations of the previously discussed modifications with sixth-order differences.

Second, we consider the combination of pressure stabilization and higher weighting of the mass residual with sixth-order differences. For a weight of $\omega_{\text{Mass}} = 1$, there is no significant improvement over either the second-order differences with pressure stabilization or the sixth-order differences without pressure stabilization. However, the results for a weight of $\omega_{\text{Mass}} = 100$ as well as $\omega_{\text{Mass}} = 1\,000$ are significantly better than using second-order differences. This is especially true for the third geometry with a weight of $\omega_{\text{Mass}} = 100$. Here the best result with sixth-order differences is 8.77% error in u and 15.67% error in p, while the best result with second-order differences is 24.42% in u and 26.87% in p. However, the results for $\omega_{\text{Mass}} = 100$ on the other two geometries are slightly worse compared to sixth-order differences without pressure stabilization.

Third, we consider the combination of the hybrid upwind scheme and higher weighting of the mass residual with sixth-order differences. With sixth-order central differences, we use third-order forward and backward differences in the hybrid scheme for the one-sided approximation of the derivatives, since these have the same stencil size in the considered direction. Compared to the results with second-order differences, hybrid upwind, and equal weighting of the lot terms, it appears that the results are worse on the first geometry, potentially better on the second geometry, and still no useful prediction has been made on the third geometry. Here we ignore results with errors so large that something clearly failed in the prediction, such as unacceptably strong oscillations. However, these results are poorer than the results for sixth-order differentials without upwind and with the same weighting of the lot terms, so the application of the upwind scheme with the threshold used here did not lead to improvements on any geometry. With a stronger weighting of the mass residuals, the prediction errors decrease on all geometries, but even here the errors obtained are still higher than for the models trained without upwind, although only slightly in the case of the first geometry. Overall, we can say that the hybrid upwind scheme considered here in combination with sixth-order differentials does not seem to provide any improvements.

Finally, we consider the combination of all four combinations, i.e. the hybrid upwind scheme with pressure stabilization and sixth-order differences as well as a higher weighting of the mass residual. For equal weighting of the loss terms, i.e. $\omega_{\text{Mass}} = 1$, the results of the combination of the other three modifications are surprisingly bad. Compared to the results without the three modifications, the predictions are very poor for the first two geometries and unchanged for the third. However, with a higher weighting of the mass residuals, the errors decrease again. For the first geometry the errors are similar to the best case. For the second geometry, the errors for $\omega_{\text{Mass}} = 100$ are similar to the best case, with slightly higher errors in the pressure. For the third geometry, comparably low errors are achieved only for $\omega_{\text{Mass}} = 1000$.

The best results for the first and second geometry were obtained for a weight of $\omega_{\text{Mass}} = 100$ and without upwind scheme and pressure stabilization. For the third geometry, the best result was also obtained without upwind scheme and pressure stabilization, but for a weight of $\omega_{\text{Mass}} = 1\,000$. However, this does not necessarily imply that these combinations generally lead to the best results. Rather, it simply states that for the few results shown here, the best results were obtained with this combination. For example, the results for the third geometry with pressure stabilization are consistently lower than without pressure stabilization. However, this combination did not produce the lowest error.

In summary, in principle, the use of sixth-order differences leads to better results than the use of second order differences. The biggest impact on the quality of the prediction is the choice of the weight of the residual mass in the loss term. Pressure stabilization has a stabilizing effect on the training, but distorts the learned solution a little. The hybrid upwind scheme considered here did not have a positive influence on the quality of the predictions. A combination of the different modifications has improved the quality of the prediction only in the case of higher order differences, stronger weighting of the mass residual and the use of pressure stabilization. In the following section, the effect of these

modifications and their combinations on a model trained on a data set is considered in detail.

6.4.6 Combination of Modifications - Multiple Geometries

In this section, we investigate the effect of applying the previously discussed modifications, and combinations thereof, on the predictive ability of models trained on a data set with multiple geometries. To do this, we train a model for each combination on the channel data set. In this section we only present results for sixth-order differences. The results presented here are only partially comparable to those in section 6.2.3. While the models are trained on the same data set, there are two differences that are relevant. First, the models in section 6.2.3 were trained on 75% of the data, which corresponds to about 3 750 geometries, while the models in this section were trained on 3 500 geometries. Second, the composition of the validation data is somewhat different in this section. In particular, there are more geometries in the validation data in this section that result in a higher flow velocity. It is precisely with these geometries that the physics-aware approach has difficulty. For these reasons, the average validation error will be slightly higher in this section.

In table 6.5, we list the relative errors in velocity and pressure averaged over the validation data, as well as the averaged absolute values of the mass residual and the pixel-wise norm of the momentum residual. The averaged values are of limited use, as we noted in the previous sections on multiple geometries. For this reason, we also show in fig. 6.47 and fig. 6.48 the distribution of the relative errors in velocity and pressure, respectively, over the maximum occurring velocity. From these plots, we can see if the modifications have an impact on, for example, predictions for geometries that are problematic for models without modifications.

From the averaged values in table 6.5 we can see that a stronger weighting of the mass residual brings an improvement even in the multiple geometry case. For a weighting of

ω_{Mass}	uw.	ϵ_Δ	$\frac{\|u_{NN}-u\|}{\|u\|}$ (%)	$\frac{\|p_{NN}-p\|}{\|p\|}$ (%)	|Mass| mean	$\|\text{Mom}\|_2$ mean
1	0	0	4.79	10.41	$3.3 \cdot 10^{-2}$	$8.9 \cdot 10^{-2}$
10	0	0	3.73	7.87	$2.1 \cdot 10^{-2}$	$1.1 \cdot 10^{-1}$
100	0	0	4.52	7.92	$1.5 \cdot 10^{-2}$	$1.7 \cdot 10^{-1}$
1 000	0	0	8.42	12.87	$1.1 \cdot 10^{-2}$	$2.8 \cdot 10^{-1}$
1	1	0	5.16	11.69	$3.3 \cdot 10^{-2}$	$8.6 \cdot 10^{-2}$
10	1	0	3.83	8.31	$2.1 \cdot 10^{-2}$	$1.2 \cdot 10^{-1}$
100	1	0	4.19	7.54	$1.4 \cdot 10^{-2}$	$1.7 \cdot 10^{-1}$
1 000	1	0	7.54	11.22	$1.1 \cdot 10^{-2}$	$3.1 \cdot 10^{-1}$
1	0	10^{-4}	4.79	10.54	$3.5 \cdot 10^{-2}$	$8.7 \cdot 10^{-2}$
10	0	10^{-4}	3.83	8.42	$2.3 \cdot 10^{-2}$	$1.2 \cdot 10^{-1}$
100	0	10^{-4}	3.94	8.34	$1.6 \cdot 10^{-2}$	$2.0 \cdot 10^{-1}$
1 000	0	10^{-4}	5.59	9.90	$1.1 \cdot 10^{-2}$	$3.7 \cdot 10^{-1}$
1	1	10^{-4}	5.46	12.70	$3.8 \cdot 10^{-2}$	$8.6 \cdot 10^{-2}$
10	1	10^{-4}	4.03	9.51	$2.3 \cdot 10^{-2}$	$1.1 \cdot 10^{-1}$
100	1	10^{-4}	4.51	9.21	$1.6 \cdot 10^{-2}$	$2.1 \cdot 10^{-1}$
1 000	1	10^{-4}	5.98	10.94	$1.1 \cdot 10^{-2}$	$4.0 \cdot 10^{-1}$

Table 6.5: Performance of the physics-aware models for combinations of the previously discussed modifications with sixth-order differences on the channel data set.

$\omega_{\text{Mass}} = 1$, the errors in velocity are 4.79% and in pressure 10.41%. With a weighting of $\omega_{\text{Mass}} = 10$, the error in velocity decreases by about 1% to 3.73% and in pressure by about 2.5% to 7.87%. These are also the best results in this section. With an even stronger weighting, however, the mean errors increase again. But this does not mean that all predictions become worse for an even stronger weighting of the mass residual. In fact, for $\omega_{\text{Mass}} = 100$ and $\omega_{\text{Mass}} = 1\,000$ the errors decrease for geometries that produce a faster velocity. Unfortunately, they increase at the same time for geometry that produces a slower velocity. This behavior is particularly evident in the distribution of errors in the pressure; see the first column of fig. 6.48. This behavior is consistent with the observations we made in section 6.4.4. There we also had to weight the mass residual more heavily for the third geometry, which produces a fast flow, than for the first and second geometries in order to achieve low errors.
This observation allows us to conclude that it might be useful to train two models separately for this data set, one for smaller obstacles and one for larger obstacles. We could train the model for larger obstacles with a higher weighting of the mass residual and the model for smaller obstacles with a lower weighting. This would allow us to choose the ideal parameter for both types of generated flow patterns.

Next, we examine the effect of the hybrid upwind scheme. With equal weighting of the loss terms, the use of the hybrid upwind scheme leads to a deterioration of the error values by about 0.5% in velocity and by about 1.0% in pressure. This effect affects all geometries equally. For higher weights of the residual mass, however, the effect of the upwind scheme is different. For example, for $\omega_{\text{Mass}} = 10$ the results deteriorate by only 0.1% and 0.5% in velocity and pressure, respectively, and for $\omega_{\text{Mass}} = 100$ and $\omega_{\text{Mass}} = 1\,000$ the results are better with upwind than without. However, these improvements are relative because overall we still get the best results with upwind for a weight of $\omega_{\text{Mass}} = 10$.

Third, we examine the effect of the pressure stabilization, here with $\epsilon = 10^{-4}$. For the first two weights considered, $\omega_{\text{Mass}} = 1$ and $\omega_{\text{Mass}} = 10$, the inclusion of pressure

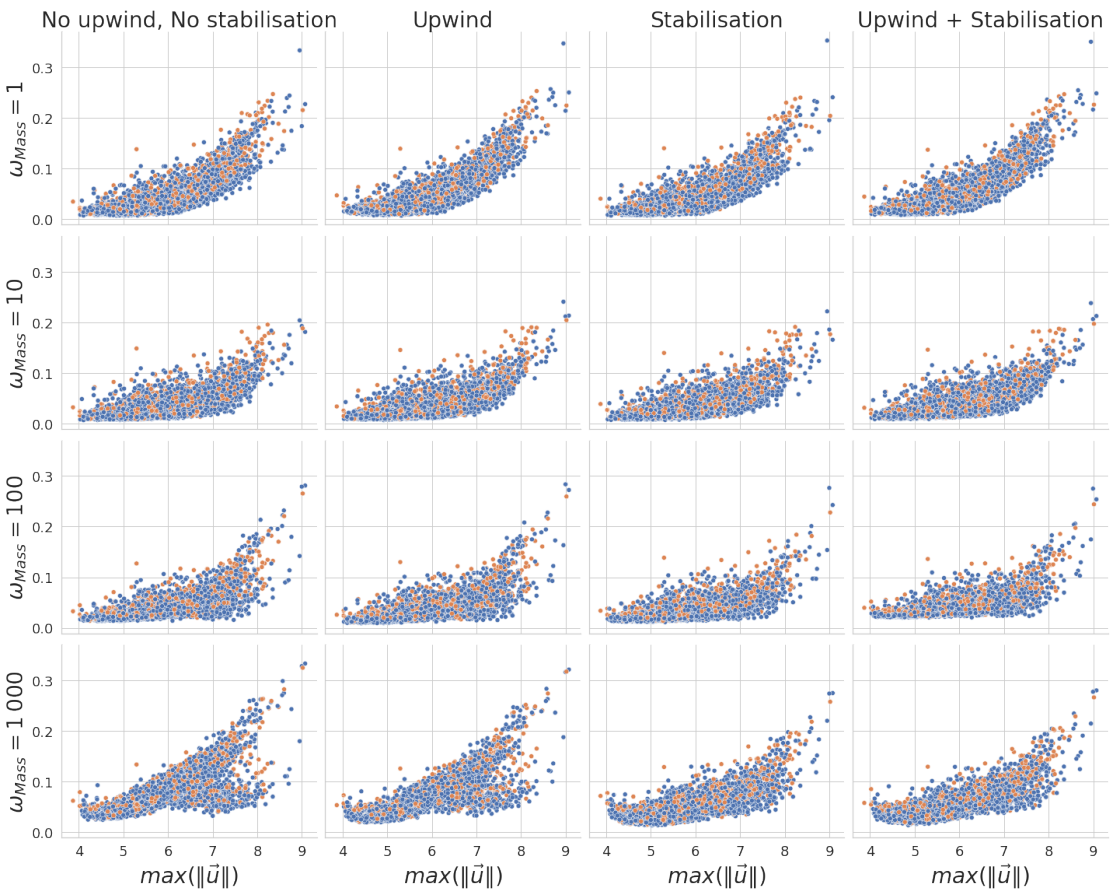


Figure 6.47: Distribution of velocity errors for physics-based models for combinations of the previously discussed modifications with sixth-order differences. Blue dots are training geometries and orange dots are validation geometries.

stabilization does not change the prediction errors, neither positively nor negatively. Only for larger weights, i.e. $\omega_{\text{Mass}} = 100$ and $\omega_{\text{Mass}} = 1\,000$, does the inclusion of pressure stabilization lead to an improvement, though not by much. For example, for $\omega_{\text{Mass}} = 100$ the error in u decreases by 0.3% while it increases in p by 0.4%. For $\omega_{\text{Mass}} = 1\,000$, however, the pressure stabilization causes the errors not to increase as much as for the model without pressure stabilization, especially for geometries that induce a flow with

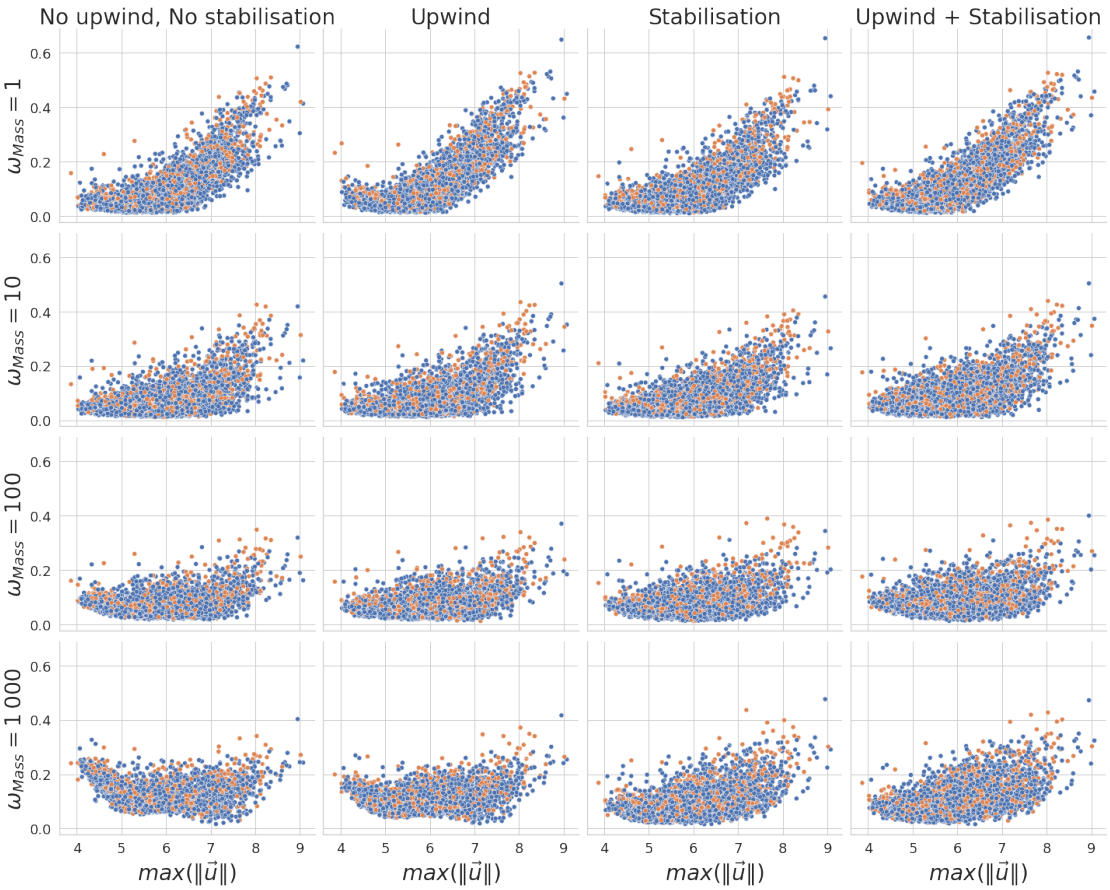


Figure 6.48: Distribution of pressure errors for physics-based models for combinations of the previously discussed modifications with sixth-order differences. Blue dots are training geometries and orange dots are validation geometries.

slower velocities.

The combination of the upwind scheme and pressure stabilization does not improve for any of the considered weights ω_{Mass}. In each case, either a model without upwind or without pressure stabilization or without both gives better results. This is consistent with what we observed regarding individual geometries in the previous section.

In summary, the largest effect comes from the weighting of the mass residual. With

an appropriate value, we are able to reduce the errors in the predictions averaged over all validation data by 1% for velocity and by 2.5% for pressure. Neither the application of the hybrid upwind scheme nor the addition of pressure stabilization lead to better results.

7 Results in Three Dimensions

In this chapter, we explore the extension of our approach to three dimensions. For this purpose, we consider the three-dimensional model problems that we have introduced in section 2.3.

In three dimensions, the computational effort and memory demand increase significantly. This is not only true for our approach, but also for classical CFD simulations. There, the meshes used and the systems of equations to be solved become significantly larger. In addition, the extra dimension can lead to intricate flow patterns that are impossible in two dimensions. In three dimensions, this only reinforces the demand for suitable surrogate models that already exists in two dimensions.

First, in section 7.1, we discuss the computational effort and memory requirements of our approach and the practical limitations this imposes. Then, we present results for three-dimensional channel geometries in section 7.2 and finally we present results for three-dimensional artery geometries in section 7.3.

7.1 Difficulties Faced in Three Dimensions

The increased memory requirement in three dimensions implies various problems for our approach. For efficiency reasons, it is important to have the entire data set and the model loaded on the GPU during training. However, forward and backpropagation are also performed there during training. Thus, additional information has to be stored depending on the size of the voxel images as well as, for example the number of layers

and convolutions in the CNN. If the voxel images and the CNN are large, we quickly reach the upper limit of the available memory on the GPU.

As an example, consider the three-dimensional channel geometries. They are represented by a $256 \times 128 \times 128$ voxel image. Reference values for a single geometry consist of four float-valued voxel images, the three velocity components and the pressure. Storing this requires 32 MB per voxel image; i.e., 128 MB in total. Thus, if we want to load a data set of 100 simulations into memory, we already need 12.5 GB – only for the data. If we want to employ a similar model as in the two-dimensional case, only with three-dimensional instead of two-dimensional convolutions, then we have a total of 2 340 convolutions in the encoder and 7 020 convolutions in the decoder. During the forward call, 7 020 floating-point pixel images with different resolutions are generated just by the activation of the convolutions, that is, the output of each individual convolution. In total, the memory required for all these activations is about 5.1 GB if the model operates at single precision. During training, much more memory is required in the course of the backpropagation algorithm. Consequently, together with the memory requirements of the data set, as well as the memory requirements of all libraries used and additional running software, such a model exceeds the available memory of 32 GB on an NVIDIA V100 GPU during training. The exact memory requirements that arise during a forward and backward call in training cannot be determined at this point, since this depends on which memory-saving techniques are applied by TensorFlow.

Let us note that there are techniques to reduce the memory consumption specifically geared towards deep learning. There are for example modified operations such as memory-efficient convolutions [25]. There are also domain-decomposition inspired architectures that enable training a single model on multiple GPUs [86].

This problem requirements handicap our approach in three dimensions and we are forced to use smaller, less expressive models. For the channel data set it is sufficient to halve the number of convolutions per level to stay below 32 GB memory. For geometries

that require larger voxel images, it is necessary to reduce the model even further.

Furthermore, the extension by an additional space dimension leads to an increase in the required computation time. Thus, an already reduced three-dimensional model needs ~ 4 seconds for the forward- and backward-pass of a training data point. A two-dimensional model of comparable size, on the other hand, requires only ~ 100 milliseconds. Thus, a three-dimensional model takes about 40 times as long to train as a two-dimensional model. Since it is not practical within the scope of this work to train a model for a double-digit number of weeks, we must shorten the training, either by reducing the number of epochs or the number of geometries on which we train.

The previous discussion about memory consumption again strongly underlines the need of training a surrogate model without reference data. To train a data-based surrogate model we need a large data set consisting of voxel images of the geometry and voxel images of the reference solutions. However, since a data set consisting of 100 simulations already occupies over 12 GB of memory, and such a data set is not considered large, it is not possible to load a large data set into memory on a GPU commonly used for deep learning. Instead of loading the entire data set into memory, however, it is possible to load only the currently required training data. But this technique would make training even slower than it already is. In contrast, our physics-aware approach does not require reference data during training, but only input image data, reducing the memory footprint of a large data set drastically. The additional memory required for the calculation of the approximated derivative is negligible here.

7.2 Channel Geometries

In this section we present results for a model trained on a data set consisting of three-dimensional channel geometries. This three-dimensional channel data set consists of equal parts of channel geometries with obstacles with 4, 5, 6, 7 and 8 edges. A description of these geometries as well as the construction of the obstacles can be found in section 2.3.2.

Due to the higher memory requirements and the longer training time per geometry, see section 7.1, only a few models were trained. Here we present only the results of the best models. It is important to note that this channel data set consists solely of images of the geometries, so there is no reference data available. For this reason, it is not possible to compare our physics-aware model with a data-based or combined model, as we did in section 6.2. In addition, we created another, smaller, data set for validation, with similar geometries to those of the training data set. This validation data set consists of 83 data points.

As described in the previous section, we are bounded in the choice of hyperparameters, so that the training of our model still fits into memory on a GPU, as well as the amount of data we can train on in a reasonable time. Thus, we had to employ a smaller CNN. Since depth, or number of levels, of our CNN is of great importance, cf. section 6.2, we chose a model that is 8 levels deep. To reduce the size of our model, it has only 32 convolutions per layer in the first level. The number of convolutions increases in the lower levels as described in section 4.5. This means that this model contains half the number of convolutions of the model we used in the two-dimensions for multiple geometries. It consists of 32 704 904 parameters. We then trained this model with our physics-aware approach on 100 training geometries from the channel data set. Here, we employed centered differences of fourth order and no regularization or upwind schemes. We slightly increased the weight of the divergence loss term to 10, as we found this to be beneficial in two dimensions; cf. section 6.4. Tests have also confirmed in three dimensions that a lower weight of the divergence loss term leads to lower performance. We use ReLu as the activation function and the Adam optimizer with a learning rate of 10^{-4} for training. This model was trained for 2 500 epochs.

The performance, measured in the relative L_2-norm, of this model evaluated on the validation data set is 3.3% in the velocity and 10.8% in the pressure. Considering the small number of training geometries we used, these errors are remarkably low. When

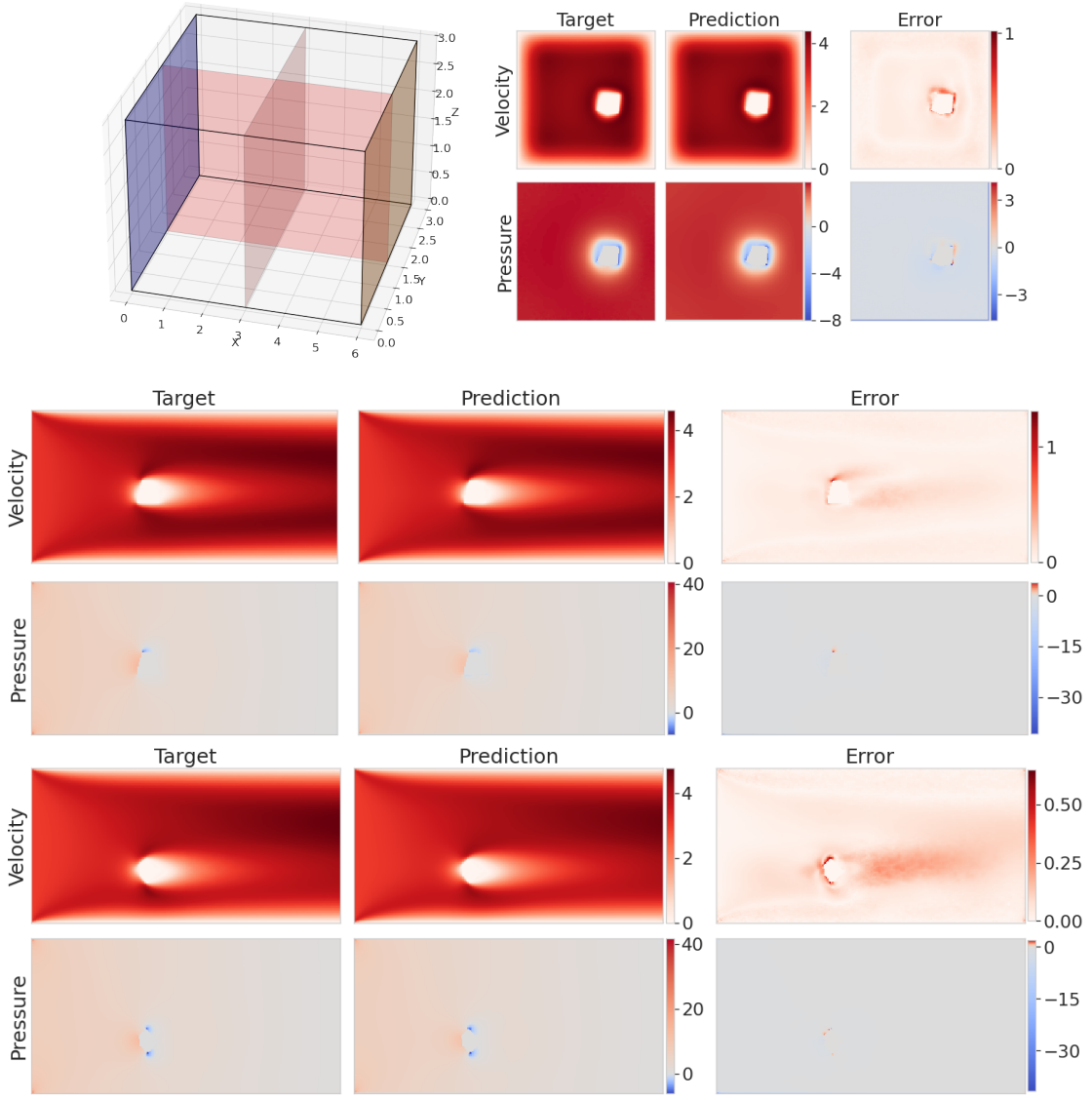


Figure 7.1: The best prediction. Target, prediction, and absolute error for velocity and pressure. The relative L_2-error in u is 1.6% and 6.7% in p. The prediction is compared to the reference solution on slices, that cut through the center of each obstacle. We show slices in the yz- plane (top right), the xz- plane (middle), and the xy- plane (bottom). We also show the yz- and xz- plane for reference (top left).

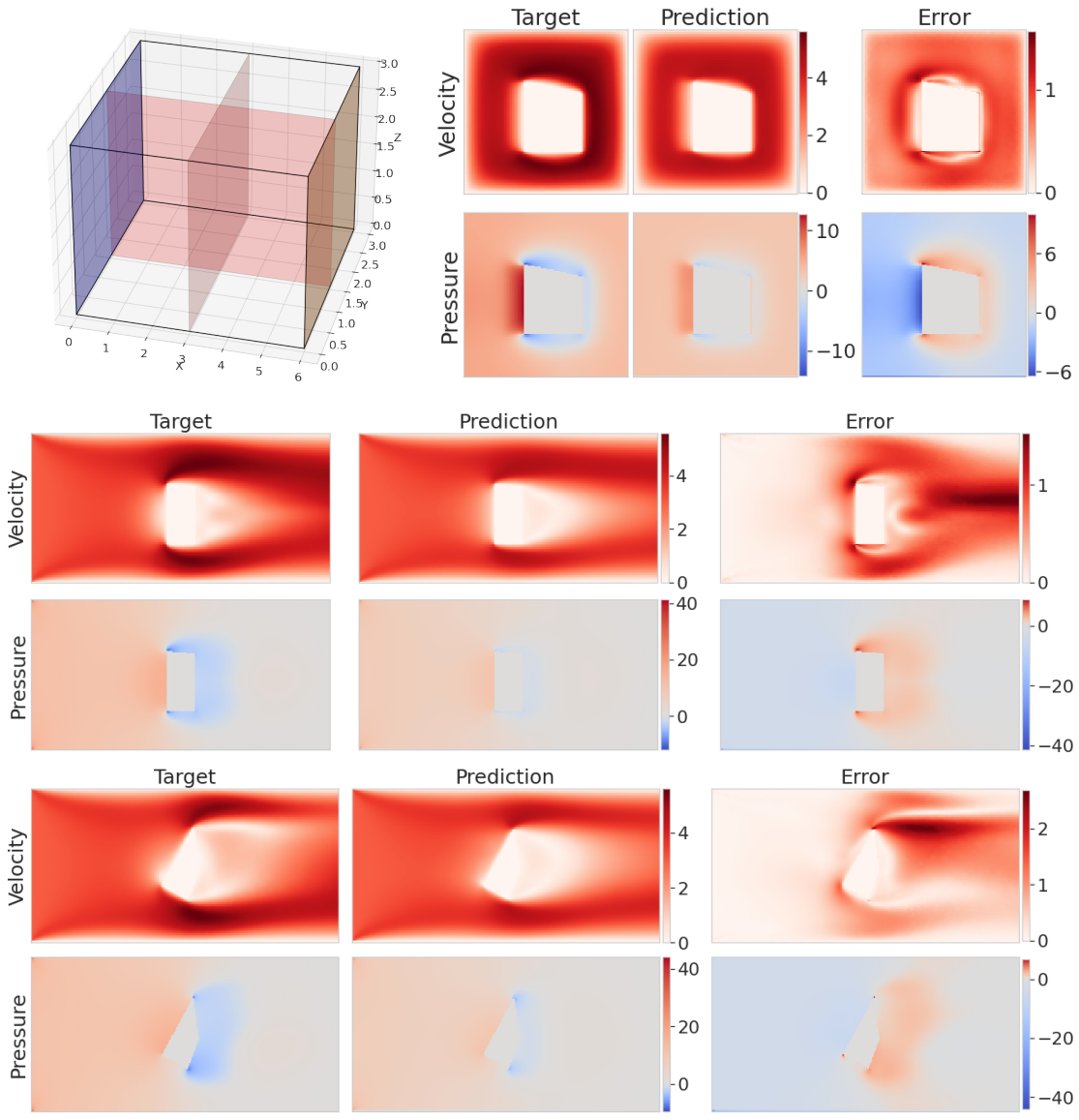


Figure 7.2: The worst prediction. Target, prediction, and absolute error for velocity and pressure. The relative L_2-error in u is 16.7% and 35.8% in p. The prediction is compared to the reference solution on slices, that cut through the center of each obstacle. We show slices in the yz- plane (top right), the xz-plane (middle), and the xy- plane (bottom). We also show the yz- and xz- plane for reference (top left).

training a comparable model on only 100 geometries in two dimensions, we obtain errors of over 20% in the velocity and over 45% in the pressure, and the predictions deviate significantly from the reference solutions. That the predictions are more accurate here can be seen in fig. 7.1, where one of the best predictions from the validation data set is shown. For this geometry, the prediction agrees very well with the reference solution. The relative error in the velocity is only 1.6% and in the pressure 6.7%. There are however also much worse predictions, as shown in fig. 7.2. The prediction on this geometry is the worst from the validation data set, with the errors being 16.7% in u and 35.8% in p. That our model does not generalize perfectly to the validation data is not surprising, however, since we trained it on only 100 geometries. Additionally, this geometry has a rather large obstacle blocking a large area in the cross section of the channel. This type of obstacle is not strongly represented in our training data set.

It should be noted that the error values presented here cannot be directly compared to the error values obtained with our models for the two-dimensional channel data set. The underlying model problem is indeed related, since the three-dimensional one is based on the two-dimensional one. However, the obstacles of the geometries contained in the data sets are of different types, and thus the corresponding reference solutions are also different.

In this section, we have shown that we can extend our approach to the third space dimension. A model trained on only a few geometries can provide good and reasonable predictions for previously unseen geometries, as long as the geometry is reasonably close to the training data.

7.3 Artery Geometries

In this section, we present results for a model trained on a data set consisting of three-dimensional single artery geometries. A detailed description of these geometries can be found in section 2.3.3.3. Specifically, this data set consists of arteries with and without

fusiform aneurysms. Due to the high memory requirements and long training times that we discussed in section 7.1, only the result for one model is shown here. As with the three-dimensional channel geometries, no reference data are available for the training data. Therefore, we evaluate the performance of our model on a separate validation data set consisting of 69 geometries.

In order to adequately represent the geometries by pixel images, we have increased the size of the images compared to the channel data set. Here, the images are $256 \times 192 \times 128$ pixels in size. This corresponds to an increase in size of 50%. As a result, the memory requirements of the model during training are higher due to the larger activation of each layer, so that we could only train a smaller model. Therefore, our model consists of 25 convolutions per layer in the first level. At the same time, we had to reduce the number of convolutions in the deeper layers, so that this model consists of 11 053 956 parameters in total. This means that the model considered here has only about a third of the parameters of the model considered for the three-dimensional channel geometries; see section 7.2. Therefore, it is to be expected that the results shown here are not optimal. Rather, they should be seen as another proof of concept that our method is capable of dealing with non-orthogonal geometries in three dimensions.

We train this model on 100 geometries from the arterial data set. To approximate the partial derivatives for the residual calculation, we use sixth-order finite differences, no regularization, and no upwind scheme. We weight the loss terms equally since we could not perform further tests on this data set due to time constraints. Furthermore, we use ReLu as the activation function and the Adam optimizer with a learning rate of 10^{-4} for training. The model was trained for 2 500 epochs.

The performance, measured in the relative L_2-norm, of this model evaluated on the validation data set is 11.3% in the velocity and 65.8% in the pressure. These errors are rather high. This is not surprising, however, since we were only able to train on 100 geometries for 2 500 epochs due to time constraints, and only on a significantly reduced

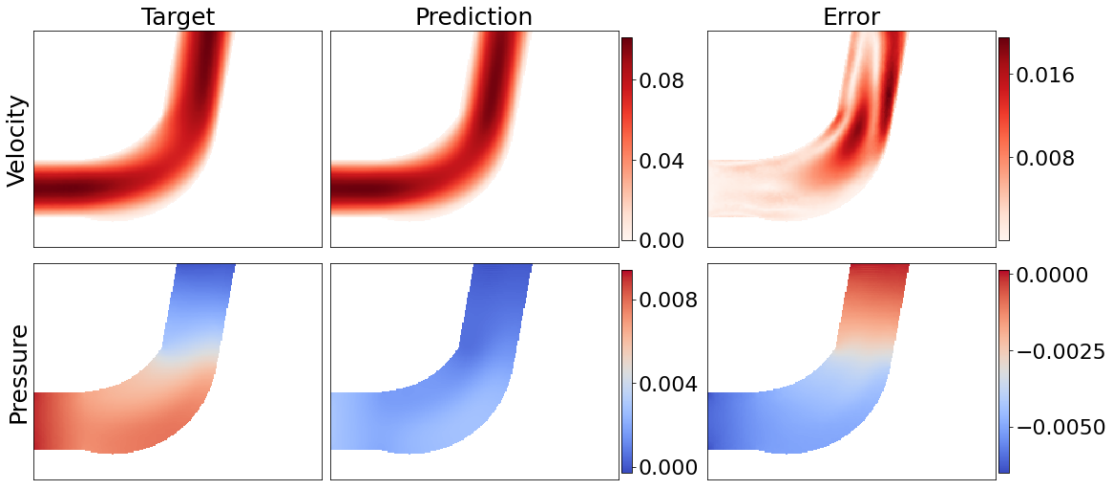


Figure 7.3: The best prediction. The relative L_2 error in u is 6.0% and 63.0% in p. We show a slice in the xy- plane along the center of the artery.

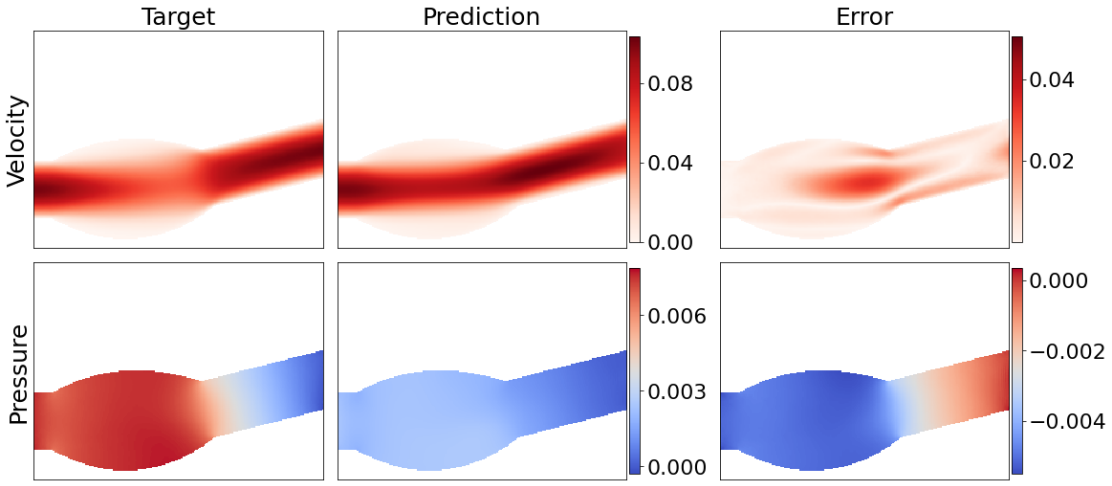


Figure 7.4: The worst prediction. The relative L_2 error in u is 19.2% in u and 75.6% in p. We show a slice in the xy- plane along the center of the artery.

model due to the larger images. These limitations appear to have severely impacted the quality of the predicted pressure. However, this is not due to potential oscillations, but due to a failure to learn correct pressure gradients. This can be seen in fig. 7.3 and fig. 7.4,

where we show the best and worst prediction over the validation data set. The errors in the velocity are 6.0% for the best and 19.2% for the worst prediction, and in the pressure 63.0% and 75.6%.

Even with only 100 training geometries, our model is capable of predicting a velocity field fairly well, as can be seen with the best prediction. Here one location where the error is large is along the straight outflow part of the artery. However, this is no longer the case when the geometries approach the extremes present in our training data set, as can be seen with the worst prediction. The geometry here features a large fusiform aneurysm with a diameter twice that of the artery at the point of its greatest extent and a length (along the center line) of more than twice the diameter of the healthy artery. The predicted velocity for this geometry deviates significantly from the reference simulation within the aneurysm. In particular, there are high errors in the velocity in the posterior region of the aneurysm.

In addition to the velocity and pressure predictions, fig. 7.5 shows the residuals of the governing equations for the predictions. Note that the values of the residuals are quite high for both the best and worst predictions. In comparison, our models for two-dimensional arterial geometries yield much lower residuals, see section 6.2.4 and in particular fig. 6.22. In particular, the mass residual is locally high, e.g., in the straight outflow of the artery of the best prediction and in the posterior part of the aneurysm of the geometry of the worst prediction. The momentum residual is also very high, especially in the areas where the error in velocity prediction is greatest.

The high errors in the prediction as well as the high residuals indicate that this model is not as well trained as the two-dimensional models. However, whether we can improve this model by training it longer, training it on more geometries, or increasing the number of convolutions and thus the number of parameters, cannot be said at this time. It may also be necessary to modify the computation of the residuals, as we did in two dimensions. We would need further testing to evaluate this. Additionally, increasing the complexity of

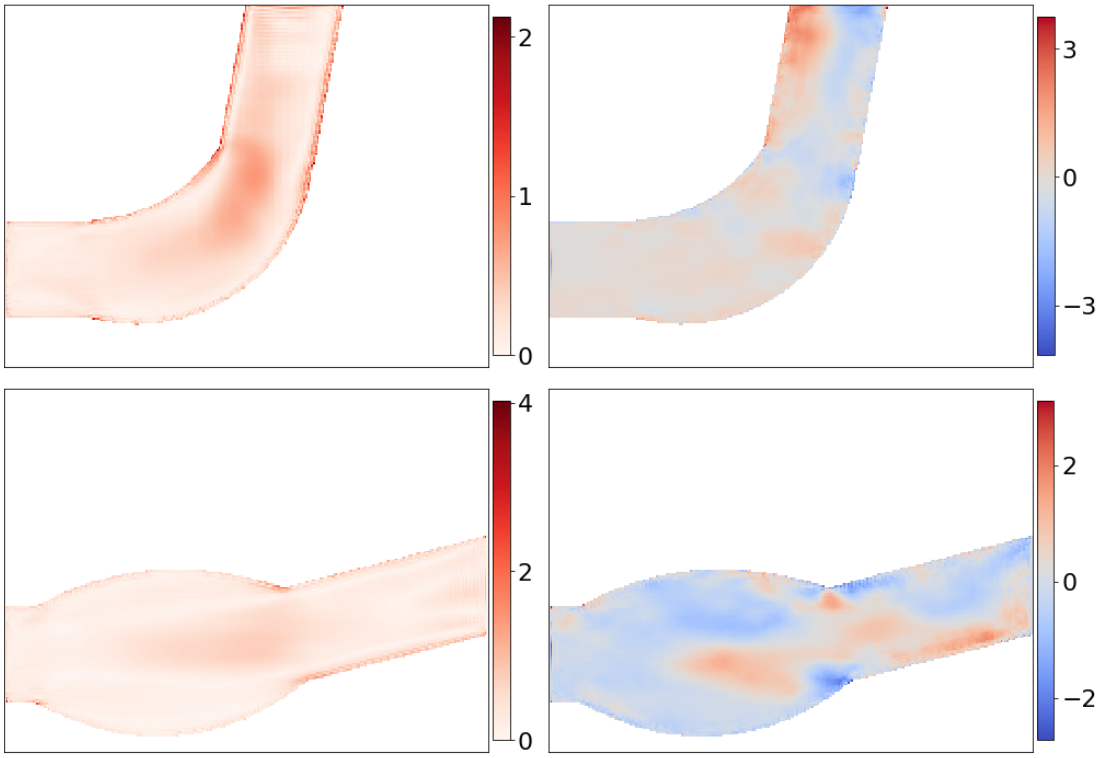


Figure 7.5: Pixelwise residuals and $\|Mom\|_2$ (left) and $Mass$ (right) for the predictions shown in fig. 7.3 (top) and fig. 7.4 (bottom). We show a slice in the xy- plane along the center of the artery.

the model, for example by increasing the number of convolutions per layer, is currently not possible due to the high memory requirements. To increase the complexity of this model, it is therefore necessary to either find a model architecture that is less memory intensive but has a higher capacity, or to find a way to train a larger model on for example multiple GPUs.

In this section, we have demonstrated that our method can be applied to non-orthogonal geometries in three dimensions. However, we had to reduce the complexity of the model significantly due to increased requirements on memory and computation time. The lower model capacity had a strong impact on the quality of the predictions, resulting in

comparatively high errors. In order to reduce these errors, it is therefore essential to work on being able to train more complex models in three dimensions.

8 Conclusion

In this thesis, we have presented and numerically tested a novel approach to train a convolutional neural network (CNN) as a surrogate model for the simulation of fluid dynamics in varying geometries using only the prior knowledge of the governing equations. The governing equations usually take the form of partial differential equations. In this work, we have explicitly applied this method to the Navier–Stokes equations for incompressible fluids. Specifically, we used the fact that the output of a CNN is a pixel image and as such can also be thought of as a grid function – or finite difference solution – on the grid of the pixel image. We use this relationship to approximate the residuals of the governing equations using finite difference stencils. We then construct a physics-aware loss function from these residuals and train our model by minimizing it. Unlike the data-based approach, this physics-aware loss function explicitly does not require labeled target data such as reference simulations. However, it is easily possible to train our approach in combination with reference data, thus combining the advantages of both approaches.

In particular, our method does not require any information about the geometry beyond the binary pixel image used as input. In this respect, our approach works for a large number of arbitrary geometries, as long as they can be represented sufficiently accurately as a pixel image. A prior parameterization of the geometries is also not required. The only additional input required is the boundary condition information. This information can be generated from the pixel image of the geometry.

However, this approach also has disadvantages. Fundamentally, a CNN can only work

with uniform grids, so we are limited to this type of grid. This is undesirable in principle, since we usually want to use higher resolutions in problematic areas without having to do so for the entire geometry. In the future, it would be interesting to overcome this limitation by, for example, using a graph neural network (GNN) instead of a CNN. Also, we are locked into one resolution once we have trained a CNN. Another significant drawback is that by representing the geometry as a pixel image, information about the geometry is lost. The amount of information lost depends on the resolution, with the loss generally being greater at coarser resolutions. In particular, straight edges that do not follow the orientation of the grid are displayed as steps.

We have performed extensive numerical experiments to confirm the strengths and weaknesses of this approach. This included comparing our physics-aware approach with the data-based approach and a combined approach consisting of training on both the physics-aware and data-based losses.

Furthermore, we have extended our approach to act as a surrogate model not only for varying geometries, but also for varying boundary conditions. This extension represents a significant challenge for contemporary physics-based machine learning algorithms, and we showed that it is straightforward for our model. In principle, our model can be extended to varying material parameters in the same way.

We have also demonstrated the ability of our approach to be used as a surrogate model on non-orthogonal geometries that mimic the shape of intracranial arteries with aneurysms. In this case, precautions had to be taken in order to prevent the occurrence of oscillations in the predictions.

Moreover, we examined a number of variants for calculating the residuals. These variants are founded on the large amount of existing work in the field of computational fluid dynamics. Thus, the effect of higher order finite difference schemes, pressure stabilization, and an upwind scheme to approximate the convective terms have been investigated. In addition, the effect of a higher weighting of the mass residual was investigated. This

proved to be particularly effective. By increasing the weighting of the mass residuals, it was possible to train a much better performing model on a wide range of geometries.

Finally, we extended our approach to three dimensions and showed in principle that it works very well on a reduced-size data set. Due to the increased memory requirements in three dimensions and present hardware limitations, we have not been able to train larger models on larger data sets. In this context, it is very interesting to consider the possibilities of memory efficient training. It may also be conceivable to distribute the model across multiple GPUs.

The main difficulty we faced in training our physics-aware models was to find an appropriate minimum. The algorithm used to minimize the loss function, the Adam optimizer, is a first-order gradient method. To obtain better results, it might be interesting to use other methods, such as the quasi-Newtonian method L-BFGS [110]. This is of particular interest because comparable physics-based ML approaches require higher-order optimizers, such as L-BFGS, to reliably produce good results [143].

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.

[2] N. Amigo, A. Valencia, W. Wu, S. Patnaik, and E. Finol. Cerebral Aneurysm Rupture Status Classification Using Statistical and Machine Learning Methods. *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, 235(6):655–662, 2021.

[3] J. D. Anderson and J. Wendt. *Computational Fluid Dynamics*, volume 206. Springer, 1995.

[4] J. O. Awoyemi, A. O. Adetunmbi, and S. A. Oluwadare. Credit Card Fraud Detection Using Machine Learning Techniques: A Comparative Analysis. In *2017 International Conference on Computing Networking and Informatics (ICCNI)*, pages 1–9. IEEE, 2017.

[5] H. Baek, M. Jayaraman, P. Richardson, and G. Karniadakis. Flow Instability and

Wall Shear Stress Variation in Intracranial Aneurysms. *Journal of the Royal Society Interface*, 7(47):967–988, 2010.

[6] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, et al. Brochure on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence. Technical report, USDOE Office of Science (SC)(United States), 2018.

[7] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic Differentiation in Machine Learning: A Survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

[8] R. M. Beam and H. E. Bailey. Newton's Method for the Navier–Stokes Equations. In *Computational Mechanics' 88: Volume 1, Volume 2, Volume 3 and Volume 4 Theory and Applications*, pages 1457–1460. Springer, 1988.

[9] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, and S. Kaushik. Prediction of Aerodynamic Flow Fields Using Convolutional Neural Networks. *Computational Mechanics*, 64:525–545, 2019.

[10] R. Biswas, M. K. Sen, V. Das, and T. Mukerji. Prestack and Poststack Inversion Using a Physics-Guided Convolutional Neural Network. *Interpretation*, 7:SE161–SE174, 2019.

[11] P. Blunsom, N. de Freitas, E. Grefenstette, and K. M. Hermann. A Deep Architecture for Semantic Parsing. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, 2014.

[12] P. B. Bochev and M. D. Gunzburger. *Least-Squares Finite Element Methods*, volume 166. Springer Science & Business Media, 2009.

[13] D. Boffi, F. Brezzi, M. Fortin, et al. *Mixed Finite Element Methods and Applications*, volume 44. Springer, 2013.

[14] S. Boyaval, C. Le Bris, T. Lelievre, Y. Maday, N. C. Nguyen, and A. T. Patera. Reduced Basis Techniques for Stochastic Problems. *Archives of Computational Methods in Engineering*, 17:435–454, 2010.

[15] D. Braess. *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer-Verlag, 2013.

[16] A. Brahme. *Comprehensive Biomedical Physics*. Newnes, 2014.

[17] E. H. Brilstra, G. J. Rinkel, Y. van der Graaf, W. J. J. van Rooij, and A. Algra. Treatment of Intracranial Aneurysms by Embolization with Coils: A Systematic Review. *Stroke*, 30(2):470–476, 1999.

[18] J. L. Brisman, J. K. Song, and D. W. Newell. Cerebral Aneurysms. *New England Journal of Medicine*, 355(9):928–939, 2006.

[19] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis. Physics-Informed Neural Networks (PINNs) for Fluid Mechanics: A Review. *Acta Mechanica Sinica*, 37(12):1727–1738, 2021.

[20] G. Calzolari and W. Liu. Deep Learning to Replace, Improve, or Aid CFD Analysis in Built Environment Applications: A Review. *Building and Environment*, 206:108315, 2021.

[21] F. Cao, F. Gao, X. Guo, and D. Yuan. Physics-Informed Neural Networks with Parameter Asymptotic Strategy for Learning Singularly Perturbed Convection-Dominated Problem. *Available at SSRN 4359807*, 2023.

[22] Q. Cao, S. Goswami, G. E. Karniadakis, and S. Chakraborty. Deep Neural Operators

Can Predict the Real-Time Response of Floating Offshore Structures Under Irregular Waves. *arXiv preprint arXiv:2302.06667*, 2023.

[23] L. Caretto, R. Curr, and D. Spalding. Two Numerical Methods for Three-Dimensional Boundary Layers. *Computer Methods in Applied Mechanics and Engineering*, 1(1):39–57, 1972.

[24] Y. Chen, F. Shi, A. G. Christodoulou, Y. Xie, Z. Zhou, and D. Li. Efficient and Accurate MRI Super-resolution Using a Generative Adversarial Network and 3D Multi-Level Densely Connected Network. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2018: 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part I*, pages 91–99. Springer, 2018.

[25] M. Cho and D. Brand. MEC: Memory-Efficient Convolution for Deep Neural Network. In *International Conference on Machine Learning*, pages 815–824. PMLR, 2017.

[26] M. Chu and N. Thuerey. Data-Driven Synthesis of Smoke Flows with CNN-Based Feature Descriptors. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017.

[27] D. Ciregan, U. Meier, and J. Schmidhuber. Multi-Column Deep Neural Networks for Image Classification. In *Conference on Computer Vision and Pattern Recognition*, pages 3642–3649. IEEE, 2012.

[28] R. Clough. The Finite Element Method in Plane Stress Analysis. In *Proc., ASME Conference on Electronic Computation, Pittsburgh, PA*, 1960.

[29] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20:273–297, 1995.

[30] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific Machine Learning Through Physics–Informed Neural Networks: Where We Are and What's Next. *Journal of Scientific Computing*, 92(3):88, 2022.

[31] M. Darwish and F. Moukalled. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab®*. Springer, 2016.

[32] V. J. DiMaio and D. K. Molina. *DiMaio's Forensic Pathology*. CRC press, 2021.

[33] M. Dissanayake and N. Phan-Thien. Neural-Network-Based Approximations for Solving Partial Differential Equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

[34] V. Dumoulin and F. Visin. A Guide to Convolution Arithmetic for Deep Learning. *arXiv preprint arXiv:1603.07285*, 2016.

[35] M. Eichinger, A. Heinlein, and A. Klawonn. Stationary Flow Predictions Using Convolutional Neural Networks. In F. J. Vermolen and C. Vuik, editors, *Numerical Mathematics and Advanced Applications ENUMATH 2019*, pages 541–549, Cham, 2021. Springer International Publishing.

[36] M. Eichinger, A. Heinlein, and A. Klawonn. Surrogate convolutional neural network models for steady computational fluid dynamics simulations. *Electronic Transactions on Numerical Analysis*, 56:235–255, 2022.

[37] M. S. Engelman and M.-A. Jamnia. Transient Flow Past a Circular Cylinder: A Benchmark Solution. *International Journal for Numerical Methods in Fluids*, 11(7):985–1000, 1990.

[38] Z. Fang. A High-Efficient Hybrid Physics-Informed Neural Networks Based on

Convolutional Neural Network. *IEEE Transactions on Neural Networks and Learning Systems*, 33(10):5514–5526, 2021.

[39] S. A. Faroughi, N. Pawar, C. Fernandes, S. Das, N. K. Kalantari, and S. K. Mahjour. Physics-Guided, Physics-Informed, and Physics-Encoded Neural Networks in Scientific Computing. *arXiv preprint arXiv:2211.07377*, 2022.

[40] J. H. Ferziger, M. Perić, and R. L. Street. *Computational Methods for Fluid Dynamics*, volume 3. Springer, 2002.

[41] B. Fornberg. Generation of Finite Difference Formulas on Arbitrarily Spaced Grids. *Mathematics of computation*, 51(184):699–706, 1988.

[42] S. Fresca, L. Dede', and A. Manzoni. A Comprehensive Deep Learning-Based Approach to Reduced Order Modeling of Nonlinear Time-Dependent Parametrized PDEs. *Journal of Scientific Computing*, 87:1–36, 2021.

[43] H. Gao, L. Sun, and J.-X. Wang. PhyGeoNet: Physics-Informed Geometry-Adaptive Convolutional Neural Networks for Solving Parameterized Steady-State PDEs on Irregular Domain. *Journal of Computational Physics*, 428:110079, 2021.

[44] H. Gao, L. Sun, and J.-X. Wang. Super-resolution and Denoising of Fluid Flow Using Physics-Informed Convolutional Neural Networks Without High-Resolution Labels. *Physics of Fluids*, 33(7):073603, 2021.

[45] H. Gao, H. Yuan, Z. Wang, and S. Ji. Pixel Transposed Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(5):1218–1227, 2019.

[46] C. Geuzaine and J.-F. Remacle. Gmsh: A Three-Dimensional Finite Element Mesh Generator with Built-In Pre- and Post-processing Facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[47] H. Gibo, C. C. Carver, A. L. Rhoton, C. Lenkey, and R. J. Mitchell. Microsurgical Anatomy of the Middle Cerebral Artery. *Journal of Neurosurgery*, 54(2):151–169, 1981.

[48] D. Giese, A. Heinlein, A. Klawonn, J. Knepper, and K. Sonnabend. Comparison of MRI Measurements and CFD Simulations of Hemodynamics in Intracranial Aneurysms Using a 3D Printed Model-Influence of Noisy MRI Measurements. *PAMM*, 19(1):e201900401, 2019.

[49] X. Glorot and Y. Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[50] X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[51] R. González-García, R. Rico-Martìnez, and I. G. Kevrekidis. Identification of Distributed Parameter Systems: A Neural Net Based Approach. *Computers & Chemical Engineering*, 22:S965–S968, 1998.

[52] I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning (Adaptive Computation and Machine Learning Series). *Cambridge Massachusetts*, pages 321–359, 2017.

[53] P. M. Gresho. Incompressible Fluid Dynamics: Some Fundamental Formulation Issues. *Annual Review of Fluid Mechanics*, 23(1):413–453, 1991.

[54] R. Grimm, M. Pettinato, S. Gillis, and W. Daelemans. Simulating Speech Processing

with Cochlear Implants: How Does Channel Interaction Affect Learning in Neural Networks? *Plos one*, 14(2):e0212134, 2019.

[55] V. Grimm, A. Heinlein, and A. Klawonn. Physics-Aware Convolutional Neural Networks for Two-Dimensional Flow Predictions. in preparation.

[56] B. A. Gross and K. U. Frerichs. Stent Usage in the Treatment of Intracranial Aneurysms: Past, Present and Future. *Journal of Neurology, Neurosurgery & Psychiatry*, 84(3):244–253, 2013.

[57] J. Grus. *Data Science from Scratch: First Principles with Python*. O'Reilly Media, 2019.

[58] X. Guo, W. Li, and F. Iorio. Convolutional Neural Networks for Steady Flow Approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 481–490, 2016.

[59] C. Haberland. *Clinical Neuropathology: Text and Color Atlas*. Demos Medical Publishing, 2006.

[60] B. Hanin. Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients? *Advances in Neural Information Processing Systems*, 31, 2018.

[61] F. H. Harlow and J. E. Welch. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *The Physics of Fluids*, 8(12):2182–2189, 1965.

[62] S. Haykin. *Neural Networks and Learning Machines*. Pearson Education India, 3 edition, 2009.

[63] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep Into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[64] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[65] Q. He, D. Barajas-Solano, G. Tartakovsky, and A. M. Tartakovsky. Physics–Informed Neural Networks for Multiphysics Data Assimilation with Application to Subsurface Transport. *Advances in Water Resources*, 141:103610, 2020.

[66] D. Hendrycks and K. Gimpel. Gaussian Error Linear Units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

[67] O. Hennigh. Lat-Net: Compressing Lattice Boltzmann Flow Simulations Using Deep Neural Networks. *arXiv preprint arXiv:1705.09036*, 2017.

[68] M. W. Hess, A. Quaini, and G. Rozza. Reduced Basis Model Order Reduction for Navier–Stokes Equations in Domains with Walls of Varying Curvature. *International Journal of Computational Fluid Dynamics*, 34(2):119–126, 2020.

[69] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[70] C. Hirt. An Arbitrary Lagrangean-Eulerian Computing Method for All Flow Speeds. *Journal of Computational Physics*, 14:203, 1974.

[71] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[72] K. Hornik, M. Stinchcombe, and H. White. Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks*, 2(5):359–366, 1989.

[73] O. J. Hénaff and E. P. Simoncelli. Geodesics of Learned Representations. *arXiv*, 2016.

[74] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456. pmlr, 2015.

[75] A. K. Jain. Data Clustering: 50 Years Beyond K-Means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[76] Y.-G. Jeong, Y.-T. Jung, M.-S. Kim, C.-K. Eun, and S.-H. Jang. Size and Location of Ruptured Intracranial Aneurysms. *Journal of Korean Neurosurgical Society*, 45(1):11, 2009.

[77] X. Jin, P. Cheng, W.-L. Chen, and H. Li. Prediction Model of Velocity Field Around Circular Cylinder over Various Reynolds Numbers by Fusion Convolutional Neural Networks Based on Pressure on the Cylinder. *Physics of Fluids*, 30(4):047105, 2018.

[78] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A Convolutional Neural Network for Modelling Sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[79] A. Kashefi and T. Mukerji. Physics-Informed PointNet: A Deep Learning Solver for Steady-State Incompressible Flows and Thermal Fields on Multiple Sets of Irregular Geometries. *Journal of Computational Physics*, 468:111510, 2022.

[80] K. Kashinath, M. Mustafa, A. Albert, J. Wu, C. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, R. Wang, A. Chattopadhyay, A. Singh, et al. Physics-Informed Machine Learning: Case Studies for Weather and Climate Modelling. *Philosophical Transactions of the Royal Society A*, 379(2194):20200093, 2021.

[81] D. Kelshaw, G. Rigas, and L. Magri. Physics-Informed CNNs for Super-resolution

of Sparse Observations on Dynamical Systems. *arXiv preprint arXiv:2210.17319*, 2022.

[82] G. Kerschen, J.-c. Golinval, A. F. Vakakis, and L. A. Bergman. The Method of Proper Orthogonal Decomposition for Dynamical Characterization and Order Reduction of Mechanical Systems: An Overview. *Nonlinear Dynamics*, 41:147–169, 2005.

[83] N. Ketkar and E. Santana. *Deep Learning with Python*, volume 1. Springer, 2017.

[84] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. In *Computer Graphics Forum*, volume 38, pages 59–70. Wiley Online Library, 2019.

[85] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[86] A. Klawonn, M. Lanser, and J. Weber. A Domain Decomposition-Based CNN-DNN Architecture for Model Parallel Training Applied to Image Recognition Problems. *arXiv preprint arXiv:2302.06564*, 2023.

[87] I. Kononenko. Machine Learning for Medical Diagnosis: History, State of the Art and Perspective. *Artificial Intelligence in Medicine*, 23(1):89–109, 2001.

[88] N. Koonjoo, B. Zhu, G. C. Bagnall, D. Bhutto, and M. Rosen. Boosting the Signal-to-Noise of Low-Field MRI with Deep Learning Image Reconstruction. *Scientific Reports*, 11(1):1–16, 2021.

[89] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and

A. Anandkumar. Neural Operator: Learning Maps Between Function Spaces. *arXiv preprint arXiv:2108.08481*, 2021.

[90] A. Krishnamurthy, S. R. Nayak, I. B. Bagoji, S. D'Costa, M. M. Pai, P. Jiji, C. Kumar, and R. Rai. Morphometry of A1 Segment of the Anterior Cerebral Artery and Its Clinical Importance. *La Clinica Terapeutica*, 161(3):231–234, 2010.

[91] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. *Communications of the ACM*, 60(6):84–90, 2017.

[92] D. Kuzmin and J. Hämäläinen. Finite Element Methods for Computational Fluid Dynamics: A Practical Guide. *SIAM Rev*, 57(4):642, 2015.

[93] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial Neural Networks for Solving Ordinary and Partial Differential Equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

[94] Y.-d. Lang, A. Malacina, L. T. Biegler, S. Munteanu, J. I. Madsen, and S. E. Zitney. Reduced Order Model Based on Principal Component Analysis for Process Simulation and Optimization. *Energy & Fuels*, 23(3):1695–1706, 2009.

[95] H. P. Langtangen, K.-A. Mardal, and R. Winther. Numerical Methods for Incompressible Viscous Flow. *Advances in Water Resources*, 25(8-12):1125–1146, 2002.

[96] O. Le Maître and O. M. Knio. *Spectral Methods for Uncertainty Quantification: With Applications to Computational Fluid Dynamics*. Springer Science & Business Media, 2010.

[97] Y. LeCun et al. Generalization and Network Design Strategies. *Connectionism in Perspective*, 19(143-155):18, 1989.

[98] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional Networks and Applications in Vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256. IEEE, 2010.

[99] K. Lee and K. T. Carlberg. Model Reduction of Dynamical Systems on Nonlinear Manifolds Using Deep Convolutional Autoencoders. *Journal of Computational Physics*, 404:108973, 2020.

[100] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila. Noise2Noise: Learning Image Restoration Without Clean Data. *arXiv preprint arXiv:1803.04189*, 2018.

[101] R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics, 2007.

[102] R. J. LeVeque et al. *Finite Volume Methods for Hyperbolic Problems*, volume 31. Cambridge university press, 2002.

[103] H. Li, X. Ren, H. Ji, C. Li, J. Wang, and J. Chen. Using Multigrid Methods in CFD Simulations. In *2016 International Conference on High Performance Computing & Simulation (HPCS)*, pages 955–960. IEEE, 2016.

[104] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the Loss Landscape of Neural Nets. *Advances in Neural Information Processing Systems*, 31, 2018.

[105] J. L. Li, X. Lin, and Z. Chen. *Finite Volume Methods for the Incompressible Navier–Stokes Equations*. Springer, 2022.

[106] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar. Physics-Informed Neural Operator for Learning Partial Differential Equations. *arXiv preprint arXiv:2111.03794*, 2021.

[107] L. Liang, D. A. Steinman, O. Brina, C. Chnafa, N. M. Cancelliere, and V. M. Pereira. Towards the Clinical Utility of CFD for Assessment of Intracranial Aneurysm Rupture–A Systematic Review and Novel Parameter-Ranking Tool. *Journal of neurointerventional surgery*, 11(2):153–158, 2019.

[108] M. Lienen and S. Günnemann. Learning the Dynamics of Physical Systems from Sparse Observations with Finite Element Networks. *arXiv preprint arXiv:2203.08852*, 2022.

[109] J. Ling, A. Kurzawski, and J. Templeton. Reynolds Averaged Turbulence Modelling Using Deep Neural Networks with Embedded Invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

[110] D. C. Liu and J. Nocedal. On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical Programming*, 45(1-3):503–528, 1989.

[111] X. Liu, J. Zhang, K. D. Nielsen, and Y. A. Cataño-Lopera. Challenges and Opportunities of Computational Fluid Dynamics in Water, Wastewater, and Stormwater Treatment. *Journal of Environmental Engineering*, 146(11):02520002, 2020.

[112] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[113] Z. Long, Y. Lu, and B. Dong. PDE-Net 2.0: Learning PDEs from Data with a Numeric-Symbolic Hybrid Deep Network. *Journal of Computational Physics*, 399:108925, 2019.

[114] Z. Long, Y. Lu, X. Ma, and B. Dong. PDE-Net: Learning PDEs from Data. In *International Conference on Machine Learning*, pages 3208–3216. PMLR, 2018.

[115] Q. Lou, X. Meng, and G. E. Karniadakis. Physics-Informed Neural Networks for Solving Forward and Inverse Flow Problems via the Boltzmann-BGK Formulation. *Journal of Computational Physics*, 447:110676, 2021.

[116] L. Lu, P. Jin, and G. E. Karniadakis. DeepONet: Learning Nonlinear Operators for Identifying Differential Equations Based on the Universal Approximation Theorem of Operators. *arXiv preprint arXiv:1910.03193*, 2019.

[117] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis. Dying ReLU and Initialization: Theory and Numerical Examples. *arXiv preprint arXiv:1903.06733*, 2019.

[118] H. Ma, Y. Zhang, N. Thuerey, X. Hu, and O. J. Haidn. Physics-Driven Learning of the Steady Navier–Stokes Equations Using Deep Convolutional Neural Networks. *arXiv preprint arXiv:2106.09301*, 2021.

[119] W. Maass. Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Neural networks*, 10(9):1659–1671, 1997.

[120] A. Malhotra, X. Wu, H. P. Forman, H. K. Grossetta Nardini, C. C. Matouk, D. Gandhi, C. Moore, and P. Sanelli. Growth and Rupture Risk of Small Unruptured Intracranial Aneurysms: A Systematic Review. *Annals of Internal Medicine*, 167(1):26–33, 2017.

[121] R. Maulik, B. Lusch, and P. Balaprakash. Reduced-Order Modeling of Advection-Dominated Systems with Recurrent Neural Networks and Convolutional Autoencoders. *Physics of Fluids*, 33(3):037106, 2021.

[122] P. W. McDonald. *The Computation of Transonic Flow Through Two-Dimensional Gas Turbine Cascades*, volume 79825. American Society of Mechanical Engineers, 1971.

[123] J. McDonough. Lectures in Computational Fluid Dynamics of Incompressible Flow: Mathematics. *Algorithms and Implementations*, 66, 2007.

[124] A. Meister. *Numerik Linearer Gleichungssysteme*, volume 4. Springer, 2011.

[125] C. Meng, S. Seo, D. Cao, S. Griesemer, and Y. Liu. When Physics Meets Machine Learning: A Survey of Physics-Informed Machine Learning. *arXiv preprint arXiv:2203.16797*, 2022.

[126] R. E. Mickens. *Difference Equations: Theory, Applications and Advanced Topics*. CRC Press, 2015.

[127] M. Milano and P. Koumoutsakos. Neural Network Modeling for Near Wall Turbulent Flow. *Journal of Computational Physics*, 182(1):1–26, 2002.

[128] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos. Image Segmentation Using Deep Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542, 2021.

[129] T. M. Mitchell and T. M. Mitchell. *Machine Learning*, volume 1. McGraw-hill New York, 1997.

[130] P. Moser, W. Fenz, S. Thumfart, I. Ganitzer, and M. Giretzlehner. Modeling of 3D Blood Flows with Physics-Informed Neural Networks: Comparison of Network Architectures. *Fluids*, 8(2):46, 2023.

[131] A. Müller and S. Guido. *Introduction to Machine Learning with Python*. O'Reilly, 3rd edition, 2017.

[132] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Icml*, 2010.

[133] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and Checkerboard Artifacts. *Distill*, 2016.

[134] J. Oldenburg, F. Borowski, A. Öner, K.-P. Schmitz, and M. Stiehm. Geometry Aware Physics Informed Neural Network Surrogate for Solving Navier–Stokes Equation (GAPINN). *Advanced Modeling and Simulation in Engineering Sciences*, 9(1):8, 2022.

[135] M. Ol'Shanskii and V. M. Staroverov. On Simulation of Outflow Boundary Conditions in Finite Difference Calculations for Incompressible Fluid. *International Journal for Numerical Methods in Fluids*, 33(4):499–534, 2000.

[136] D. W. Otter, J. R. Medina, and J. K. Kalita. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624, 2020.

[137] M. Pak and S. Kim. A Review of Deep Learning in Image Recognition. In *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, pages 1–3. IEEE, 2017.

[138] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*. CRC press, 2018.

[139] C.-H. Pham, A. Ducournau, R. Fablet, and F. Rousseau. Brain MRI Super-resolution Using Deep 3D Convolutional Networks. In *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*, pages 197–200. IEEE, 2017.

[140] S. B. Pope and S. B. Pope. *Turbulent Flows*. Cambridge University Press, 2000.

[141] A. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations*, volume 23. Springer Science & Business Media, 2008.

[142] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics Informed Deep Learning (Part I): Data-Driven Solutions of Nonlinear Partial Differential Equations. *arXiv preprint arXiv:1711.10561*, 2017.

[143] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations. *Journal of Computational Physics*, 378:686–707, 2019.

[144] P. V. Raja, J. Huang, A. V. Germanwala, P. Gailloud, K. P. Murphy, and R. J. Tamargo. Microsurgical Clipping and Endovascular Coiling of Intracranial Aneurysms: A Critical Review of the Literature. *Neurosurgery*, 62(6):1187–1203, 2008.

[145] P. Ramachandran, B. Zoph, and Q. Le. Searching for Activation Functions. *arXiv*, 2017.

[146] S. Ranftl, W. von der Linden, and M. . S. Committee. Bayesian Surrogate Analysis and Uncertainty Propagation. In *Physical Sciences Forum*, volume 3, page 6. MDPI, 2021.

[147] F. Regazzoni, S. Pagani, and A. Quarteroni. Universal Solution Manifold Networks (USM-Nets): Non-intrusive Mesh-Free Surrogate Models for Problems in Variable Domains. *Journal of Biomechanical Engineering*, 144(12):121004, 2022.

[148] D. Rempfer. On boundary conditions for incompressible navier–stokes problems. *Applied Mechanics Reviews*, 59:107–125, 2006.

[149] P. Ren, C. Rao, Y. Liu, J.-X. Wang, and H. Sun. PhyCRNet: Physics-Informed Convolutional-Recurrent Network for Solving Spatiotemporal PDEs. *Computer Methods in Applied Mechanics and Engineering*, 389:114399, 2022.

[150] Rhcastilhos. Schematic Representation of the Circle of Willis, Arteries of the Brain and Brain Stem, 2007.

[151] C. M. Rhie and W.-L. Chow. Numerical Study of the Turbulent Flow past an Airfoil with Trailing Edge Separation. *AIAA journal*, 21(11):1525–1532, 1983.

[152] M. D. Ribeiro, A. Rehman, S. Ahmed, and A. Dengel. DeepCFD: Efficient Steady-State Laminar Flow Approximation with Deep Convolutional Neural Networks. *arXiv preprint arXiv:2004.08826*, 2020.

[153] T. Richter. *Fluid-Structure Interactions: Models, Analysis and Finite Elements*, volume 118. Springer, 2017.

[154] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[155] R. S. Rosenson, A. McCormick, and E. F. Uretz. Distribution of Blood Viscosity Values and Biochemical Correlates in Healthy Adults. *Clinical Chemistry*, 42(8):1189–1195, 08 1996.

[156] S. Ruder. An Overview of Gradient Descent Optimization Algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[157] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Representations by Back-Propagating Errors. *nature*, 323(6088):533–536, 1986.

[158] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[159] M. Sadoughi and C. Hu. Physics-Based Convolutional Neural Network for Fault Diagnosis of Rolling Element Bearings. *IEEE Sensors Journal*, 19(11):4181–4192, 2019.

[160] J. E. Santos, D. Xu, H. Jo, C. J. Landry, M. Prodanović, and M. J. Pyrcz. PoreFlow-Net: A 3D Convolutional Neural Network to Predict Fluid Flow Through Porous Media. *Advances in Water Resources*, 138:103539, 2020.

[161] G. M. Sasidharan, S. B. Sastri, P. Pandey, et al. Aneurysm Clips: What Every Resident Should Know. *Neurology India*, 63(1):96, 2015.

[162] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[163] A. Scheinker and R. Pokharel. Physics-Constrained 3D Convolutional Neural Networks for Relativistic Electrodynamics. *Bulletin of the American Physical Society*, 2023.

[164] D. Scherer, A. Müller, and S. Behnke. Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. In *Artificial Neural Networks–ICANN 2010: 20th International Conference, Thessaloniki, Greece, September 15-18, 2010, Proceedings, Part III 20*, pages 92–101. Springer, 2010.

[165] S. J. Schreck, W. E. Faller, and M. W. Luttges. Neural Network Prediction of Three-Dimensional Unsteady Separated Flowfields. *Journal of Aircraft*, 32(1):178–185, 1995.

[166] V. Sekar, Q. Jiang, C. Shu, and B. C. Khoo. Fast Flow Field Prediction over Airfoils Using Deep Learning Approach. *Physics of Fluids*, 31(5):057103, 2019.

[167] U. Senturk, D. Brunner, H. Jasak, N. Herzog, C. W. Rowley, and A. J. Smits. Benchmark Simulations of Flow Past Rigid Bodies Using an Open-Source, Sharp Interface Immersed Boundary Method. *Progress in Computational Fluid Dynamics, an International Journal*, 19(4):205–219, 2019.

[168] R. Sharma, A. B. Farimani, J. Gomes, P. Eastman, and V. Pande. Weakly-Supervised Deep Learning of Heat Transport via Physics Informed Loss. *arXiv preprint arXiv:1807.11374*, 2018.

[169] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning Semantic Representations Using Convolutional Neural Networks for Web Search. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 373–374, 2014.

[170] G. D. Smith and G. D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford university press, 1985.

[171] C. Soize. *Uncertainty Quantification*. Springer, 2017.

[172] D. B. Spalding. A Novel Finite Difference Formulation for Differential Expressions Involving Both First and Second Derivatives. *International Journal for Numerical Methods in Engineering*, 4(4):551–559, 1972.

[173] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training Very Deep Networks. *Advances in Neural Information Processing Systems*, 28, 2015.

[174] J. C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. SIAM, 2004.

[175] N. Sukumar and A. Srivastava. Exact Imposition of Boundary Conditions with Distance Functions in Physics-Informed Deep Neural Networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114333, 2022.

[176] L. Sun, H. Gao, S. Pan, and J.-X. Wang. Surrogate Modeling for Fluid Flows Based on Physics-Constrained Deep Learning Without Simulation Data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020.

[177] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[178] A. Takbiri-Borujeni, H. Kazemi, and N. Nasrabadi. A Data-Driven Surrogate to Image-Based Flow Simulations in Porous Media. *Computers & Fluids*, 201:104475, 2020.

[179] B. Taylor. *Methodus incrementorum directa et inversa*. Innys, 1717.

[180] The OpenFOAM Foundation. *OpenFOAM v8 User Guide*.

[181] J. Thiyagalingam, M. Shankar, G. Fox, and T. Hey. Scientific Machine Learning Benchmarks. *Nature Reviews Physics*, 4(6):413–420, 2022.

[182] B. G. Thompson, R. D. Brown Jr, S. Amin-Hanjani, J. P. Broderick, K. M. Cockroft, E. S. Connolly Jr, G. R. Duckwiler, C. C. Harris, V. J. Howard, S. C. Johnston, et al. Guidelines for the Management of Patients with Unruptured Intracranial Aneurysms: A Guideline for Healthcare Professionals from the American Heart Association/American Stroke Association. *Stroke*, 46(8):2368–2400, 2015.

[183] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating Eulerian Fluid Simulation with Convolutional Networks. In *International Conference on Machine Learning*, pages 3424–3433. PMLR, 2017.

[184] B. Ummenhofer, L. Prantl, N. Thuerey, and V. Koltun. Lagrangian Fluid Simulation with Continuous Convolutions. In *International Conference on Learning Representations*, 2020.

[185] H. A. Van der Vorst. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.

[186] M. W. Vernooij, M. A. Ikram, H. L. Tanghe, A. J. Vincent, A. Hofman, G. P. Krestin, W. J. Niessen, M. M. Breteler, and A. van der Lugt. Incidental Findings on Brain MRI in the General Population. *New England Journal of Medicine*, 357(18):1821–1828, 2007.

[187] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education, 2007.

[188] I. E. Vignon-Clementel, C. A. Figueroa, K. E. Jansen, and C. A. Taylor. Outflow Boundary Conditions for Three-Dimensional Finite Element Modeling of Blood Flow and Pressure in Arteries. *Computer Methods in Applied Mechanics and Engineering*, 195(29-32):3776–3796, 2006.

[189] L. Von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, J. Pfrommer, A. Pick, R. Ramamurthy, et al. Informed Machine Learning–A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(1):614–633, 2021.

[190] R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu. Towards Physics-Informed Deep Learning for Turbulent Flow Prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1457–1466, 2020.

[191] S. Wang, H. Wang, and P. Perdikaris. Learning the Solution Operator of Parametric Partial Differential Equations with Physics-Informed DeepONets. *Science advances*, 7(40):eabi8605, 2021.

[192] M. Watanabe. *ADAN: An Human Anatomically Detailed Arterial Network for Computational Hemodynamics*. PhD thesis, National Laboratory for Scientific Computing, 2013.

[193] J. Watt, R. Borhani, and A. K. Katsaggelos. *Machine Learning Refined: Foundations, Algorithms, and Applications*. Cambridge University Press, 2020.

[194] Y. Wei. The Development and Application of CFD Technology in Mechanical Engineering. In *IOP Conference Series: Materials Science and Engineering*, volume 274, page 012012. IOP Publishing, 2017.

[195] J. F. Wendt. *Computational Fluid Dynamics: An Introduction*. Springer Science & Business Media, 2008.

[196] P. M. White, E. M. Teasdale, J. M. Wardlaw, and V. Easton. Intracranial Aneurysms: CT Angiography and MR Angiography for Detection—Prospective Blinded Comparison in a Large Patient Cohort. *Radiology*, 219(3):739–749, 2001.

[197] S. Wiewel, M. Becher, and N. Thuerey. Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow. In *Computer Graphics Forum*, volume 38, pages 71–82. Wiley Online Library, 2019.

[198] L. N. Williams and R. D. Brown. Management of Unruptured Intracranial Aneurysms. *Neurology: Clinical Practice*, 3(2):99–108, 2013.

[199] J. Wu, X. Yin, and H. Xiao. Seeing Permeability from Images: Fast Prediction with Convolutional Neural Networks. *Science Bulletin*, 63(18):1215–1222, 2018.

[200] D. Xiao, F. Fang, A. G. Buchan, C. C. Pain, I. M. Navon, and A. Muggeridge. Non-intrusive Reduced Order Modelling of the Navier–Stokes Equations. *Computer Methods in Applied Mechanics and Engineering*, 293:522–541, 2015.

[201] Y. Xu, S. Kohtz, J. Boakye, P. Gardoni, and P. Wang. Physics-Informed Machine Learning for Reliability and Systems Safety Applications: State of the Art and Challenges. *Reliability Engineering & System Safety*, page 108900, 2022.

[202] Z. Xu, Y.-N. Rui, J. P. Hagan, and D. H. Kim. Intracranial Aneurysms: Pathology, Genetics, and Molecular Mechanisms. *Neuromolecular Medicine*, 21(4):325–343, 2019.

[203] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear Spatial Pyramid Matching Using Sparse Coding for Image Classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1794–1801. IEEE, 2009.

[204] X. Ying. An Overview of Overfitting and Its Solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

[205] L. Zarrinkoob, K. Ambarki, A. Wåhlin, R. Birgander, A. Eklund, and J. Malm. Blood Flow Distribution in Cerebral Arteries. *Journal of Cerebral Blood Flow & Metabolism*, 35(4):648–654, 2015.

[206] M. D. Zeiler, G. W. Taylor, and R. Fergus. Adaptive Deconvolutional Networks for Mid and High Level Feature Learning. In *2011 International Conference on Computer Vision*, pages 2018–2025. IEEE, 2011.

[207] R. Zhang, Y. Liu, and H. Sun. Physics-Guided Convolutional Neural Network (PhyCNN) for Data-Driven Seismic Response Modeling. *Engineering Structures*, 215:110704, 2020.

[208] Z. Zhang, X. Yan, P. Liu, K. Zhang, R. Han, and S. Wang. A Physics-Informed Convolutional Neural Network for the Simulation and Prediction of Two-Phase Darcy Flows in Heterogeneous Porous Media. *Journal of Computational Physics*, page 111919, 2023.

[209] G. Zhou, Y. Zhu, Y. Yin, M. Su, and M. Li. Association of Wall Shear Stress with Intracranial Aneurysm Rupture: Systematic Review and Meta-Analysis. *Scientific Reports*, 7(1):1–8, 2017.

[210] Y. Zhu and N. Zabaras. Bayesian Deep Convolutional Encoder–Decoder Networks for Surrogate Modeling and Uncertainty Quantification. *Journal of Computational Physics*, 366:415–447, 2018.

Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel und Literatur angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind als solche kenntlich gemacht. Ich versichere an Eides statt, dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie - abgesehen von unten angegebenen Teilpublikationen und eingebundenen Artikeln und Manuskripten - noch nicht veröffentlicht worden ist sowie, dass ich eine Veröffentlichung der Dissertation vor Abschluss der Promotion nicht ohne Genehmigung des Promotionsausschusses vornehmen werde. Die Bestimmungen dieser Ordnung sind mir bekannt. Darüber hinaus erkläre ich hiermit, dass ich die Ordnung zur Sicherung guter wissenschaftlicher Praxis und zum Umgang mit wissenschaftlichem Fehlverhalten der Universität zu Köln gelesen und sie bei der Durchführung der Dissertation zugrundeliegenden Arbeiten und der schriftlich verfassten Dissertation beachtet habe und verpflichte mich hiermit, die dort genannten Vorgaben bei allen wissenschaftlichen Tätigkeiten zu beachten und umzusetzen. Ich versichere, dass die eingereichte elektronische Fassung der eingereichten Druckfassung vollständig entspricht.

Köln, 30.04.2023

Ort, Datum | Unterschrift